VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Uživatelská aplikace pro Andorid OS pro produkt testbed

User App for Android OS for Testbed

Bc. Lukáš Tatarin

Diplomová práce

Vedoucí práce: doc. Ing. Jaromír Konečný, Ph.D.

Ostrava, 2021

Abstrakt

Cílem této diplomové práce je vytvoření aplikace pro mobilní operační systém Android, která bude pomocí technologie Bluetooth Low Energy komunikovat s produktem testbed. Aplikace se připojuje k jednotlivým testbedům a ukládá jimi získaná data do lokální databáze v každém mobilním zařízení, na kterém je nainstalována. Úkolem aplikace je také zobrazit uživateli přehledně naměřená data a poskytnout mu základní přehled o naměřených hodnotách a jejich proměnách v závislosti na čase měření.

Klíčová slova

Testbed, Android OS, Bluetooth Low Energy, SQLite, Internet věcí, Java

Abstract

The aim of this master's thesis is to create an application for the Android operating system which will communicate with the testbed product using Bluetooth Low Energy technology. The application is connected to individual testbeds and stores the data obtained by them in a local database on each mobile device on which it is installed. The application can also display to the user clearly measured data and provide him with a basic overview of the measured values and their changes depending on the measurement time.

Keywords

Testbed, Android OS, Bluetooth Low Energy, SQLite, Internet of Things, Java

Poděkování

Rád bych na tomto místě poděkoval panu doc. Ing. Jaromírovi Konečnému, Ph.D. za metodické vedení mé diplomové práce a možnosti podílet se na novém testbedu Vysoké školy báňské – Technické univerzity Ostrava. Velký dík patří také mé rodině a přítelkyni, kteří mně byli vždy velkou oporou během celého studia. Mé poděkování patří také Mgr. Silvii Holé za korekturu diplomové práce.

Obsah

Seznam použitých zkratek a symbolů

3G	–	třetí generace bezdrátových sítí
5G	–	pátá generace bezdrátových sítí
ANR	–	Application Not Responding
API	–	Application Programming Interface)
BAP	–	Battery profile
BLE	–	Bluetooth Low Energy
BPM	–	Beats Per Minute
CPIT	–	Centrum pokročilých inovačních technologií
CPU	–	Central Processing Unit
CSV	–	Comma-Separated Value
DPS	–	deska plošných spojů
EEPROM	–	Electrically Erasable Programmable Read-Only Memory
ENIG	–	Electroless Nickel Immersion Gold
FHSS	–	Frequency Hopping Spread Spectrum
GAP	–	Generic Access Profile
GATT	–	Generic Attribute Profile
GSM	–	Groupe Spécial Mobile
GPS	–	Global Positioning System
HCI	–	Host Controller Interface
HRP	–	Heart Rate Profile
IEEE	–	Institute of Electrical and Electronics Engineers
ISM	–	Industrial, Scientific and Medical
ISO	–	International Organization for Standardization
IoT	–	Internet of Things
IP	–	Internet Protocol
L2CAP	–	Logical Link and Adaptation Protocol
LAN	–	Local Area Network
LCD	–	Liquid Crystal Display

LNP	–	Location and Navigation Profile
LTE	–	Long Term Evolution
LSB	–	Least Significant Bit
MAC	–	Media Access Control
MAN	–	Metropolitan Area Network
MSB	–	Most Significant Bit
OS	–	operační systém
OSI	–	Open Systems Interconnection
PAN	–	Personal Area Network
SIG	–	Special Interest Group
SQL	–	Structured Query Language
TL3	–	těžká laboratoř 3
TV	–	televize
UDP	–	User Datagram Protocol
UML	–	Unified Modeling Language
USB	–	Universal Serial Bus
UUID	–	Universally Unique Identifier
Wi-Fi	–	Wireless Fidelity
WAN	–	Wide Area Network
WLAN	–	Wireless Local Area Network

Seznam obrázků

Seznam výpisů zdrojových kódů

Úvod

Na přelomu desátých a dvacátých let 21. století stojí svět na prahu další, v pořadí již čtvrté, průmyslové revoluce. Oproti předchozím, které přinesly industrializaci, elektrifikaci nebo automatizaci, přinesla tato propojení výroby pomocí internetu a zavedení nových postupů a možností, které do té doby nebylo možné realizovat. Jako s každou průmyslovou revolucí, tak i s touto přichází zásadní proměna společenského života, která má důsledky ve změnách životní úrovně, pracovního trhu nebo vzdělávacího systému.

Na Vysoké škole báňské – Technické univerzitě Ostrava vznikl pro účely výuky a výzkumu nový testbed umístěný v nové laboratoři CPIT TL3, který v sobě kombinuje všechny prvky typické právě pro průmysl 4.0. Laboratoř má sloužit k výzkumu a také vzdělávání nových techniků a inženýrů, kteří budou v následujících letech na pracovním trhu čím dál potřebnější a nahradí méně kvalifikované pracovní síly, jejichž místo převezmou stroje.

Hlavní účel, ke kterému má testbed sloužit, je demonstrace testování sériově vyráběných elektrických zařízení a ověřování jejich správné funkčnosti před naskladněním a následnou expedicí. Produkt testbed je zařízení (logger), které zaznamenává kroky (pomocí kombinovaného akcelerometru), srdeční tep a teplotu. Tato data zobrazuje na displeji a také odesílá skrze rozhraní Bluetooth Low Energy (BLE).

Cílem této diplomové práce je návrh, implementace a testování aplikace pro mobilní operační systém Android, skrze kterou by se uživatel se svým mobilní zařízením připojil k produktu testbed. Aplikace by měla data sesbíraná produktem testbed uložit do lokální databáze a vhodným a uživatelsky příjemným způsobem je prezentovat.

První kapitola této práce se zabývá definicí průmyslu 4.0 a souvisejících technologií jako jsou internet věcí, nebo testbed. Druhá kapitola pak shrnuje informace o mobilních platformách a zaměřuje se na specifika vývoje aplikací pro operační systém Android. Třetí kapitola je věnovaná technologii Bluetooth a zejména její čtvrté verzi Bluetooth Low Energy. Čtvrtá kapitola práce se věnuje popisu samotného produktu testbed a jeho komunikačního rozhraní. Pátá kapitola popisuje návrh aplikace podle vodopádového modelu a modelování aplikace pomocí UML. V šesté kapitole je detailně popsána samotná implementace jednotlivých částí aplikace. Poslední, sedmá kapitola se věnuje testování navržené aplikace.

Kapitola 1

Testbed a další prvky průmyslu 4.0

Se čtvrtou průmyslovou revolucí přišlo zároveň mnoho nových a čím dál častěji skloňovaných pojmů. Jedním z nich jsou internet věcí a také testbed. Tyto pojmy se staly nedílnou součástí moderní společnosti 21. století, jelikož jsou stále více zastoupeny nejen v průmyslu, ale také v dopravě, zdravotnictví nebo osobním životě.

1.1 Průmyslové revoluce

První průmyslová revoluce započala v Anglii na konci 18. století. Spouštěčem byl vynález prvního mechanického tkalcovského stavu Edmundem Cartwrightem v roce 1784. [1] V té době se masově začaly nasazovat nové zdroje energie (pára), které z části nahradily lidskou sílu. Tento převrat měl obrovský celospolečenský dopad. Došlo ke kompletní změně životního stylu, zániku manufaktur a přesunu k průmyslové výrobě a také ke vzniku soukromého vlastnictví. Zjednodušeně můžeme říci, že první průmyslová revoluce vedla k masivnímu nasazování strojů do výroby. Tento jev je znám pod názvem industrializace. [1]

Druhá vlna průmyslové revoluce plynule navázala na vlnu předchozí. Koncem 19. století vznikaly ve velkých výrobních závodech a továrnách první montážní linky. Díky nim bylo možné práci nad jednotlivými produkty rozdělit a přejít k efektivnější sériové výrobě. S touto vlnou je také spjata první elektrifikace celé výroby. [1]

Třetí průmyslová revoluce byla spíše než revolucí přirozenou evolucí, kterou již nebylo možné déle zdržet. Začala v 70. letech minulého století a odstartoval ji rozmach polovodičů a s tím spjaté první počítače. Ve výrobě se začaly nasazovat první programovatelné automaty, které mohly pomocí řady senzorů a aktuátorů opět zefektivnit výrobu a celý proces do jisté míry zautomatizovat. [1]

Posledních několik málo let se kolem nás odehrává další průmyslová revoluce. Ta bývá odborníky označovaná jako čtvrtá a neodmyslitelně je s ní spjat pojem Průmysl 4.0. [1]

1.2 Průmysl 4.0

Pojem průmysl 4.0 byl poprvé představen v roce 2011 na veletrhu v Hannoveru jako součást programu německé vlády na rozsáhlou digitalizaci a využívání robotů v průmyslu. Cílem vize mělo být zvýšení konkurenceschopnosti a vytvoření dostatečného množství pracovních sil pro kvalifikované zaměstnance. Německá vláda dotovala tento program částkou 50 milionů eur po dobu 3 let a zapojila do něj přední německé strojírenské firmy. [2]

Hlavní myšlenkou čtvrté vlny průmyslové revoluce je vybudování chytrých továren – anglicky smart factory – kde dosud jednoduché a často opakující se činnosti nahradí kyberneticko-fyzikální systémy. Přímým důsledkem bude tak jako u předchozích průmyslových revolucí změna pracovního trhu, kde robotické systémy nahradí méně kvalifikovanou lidskou sílu a vytvoří nová pracovní místa, která již budou ale požadovat náročnější kvalifikaci. Jinými slovy zaniknou pracovní pozice, které výrobu přímo obsluhují a vzniknou pracovní místa, která výrobu projektují a budují a posléze udržují v chodu. [3]

Hlavními nástroji, kterými má být průmyslu 4.0 dosaženo jsou relativně čerstvé technologie z různých oblastí. Z oblasti informatiky můžeme zmínit například strojové vnímání a učení nebo cloudová úložiště. Z oblasti strojní to jsou pak například technologie 3D tisku. Neméně důležitá je i oblast telekomunikační, která zajišťuje přenos velkého množství dat pomocí internetu věcí a moderních 5G sítí. [3]

Kromě úspory času a financí při výrobě přináší čtvrtá průmyslová revoluce i naději na zvýšení kvality lidského života díky odstranění monotonních pracovních úkonů a fyzicky náročných činností. Mohla by rovněž přinést i příznivější podmínky pro zavedení základního nepodmíněného příjmu občanů. [4]

1.3 Internet věcí

Tento pojem pod anglickou zkratkou IoT (Internet of Things) nese označení pro síť koncových zařízení, která jsou pomocí síťového rozhraní připojena do celosvětové sítě internetu. Každé z těchto zařízení je schopno pracovat samostatně, vyměňovat si data s jinými zařízeními, případně jiné zařízení řídit. Takovými zařízeními nejsou přímo myšleny počítače nebo mobilní telefony, které uživatelé používají pro komunikaci nebo získávání informací pomocí internetu, ačkoliv je možné skrze ně koncové zařízení internetu věcí ovládat, ale zařízení jako jsou domácí spotřebiče, chytré senzory, automobily nebo roboty. Dle odhadů bude v roce 2025 připojeno do internetu na 75 miliard koncových zařízení. [5]

Praktická aplikace použití internetu věcí je napříč všemi odvětvími lidských činností velmi rozsáhlá. V oblasti osobního využití můžeme internet věcí najít ve spotřebitelské elektronice. Jedná se o chytré ledničky, pračky, klimatizace, svítidla a další prvky chytré domácnosti, které mají za cíl zvýšit uživatelský komfort a snížit provozní náklady. V oblasti řízení infrastruktury se pomocí

chytrých senzorů a aktuátorů zase optimalizuje provoz veřejného osvětlení, veřejné dopravy nebo odpadového hospodářství. Své využití nalézá zejména v posledních letech internet věcí i v oblasti zemědělství. Pomocí enviromentálních senzorů lze získávat data o teplotě, úhrnu srážek, rychlosti a směru větru, atmosférickém tlaku nebo půdní vlhkosti. [6]

1.3.1 Bezdrátové technologie

Koncová zařízení se připojují do sítě pomocí různých, většinou bezdrátových technologií. Pro zařízení v domácnosti je typické využití právě Wi-Fi. Tou je většina domácností pokrytá a samotná integrace takový zařízení je podstatně jednodušší. Podobně je v domácím nebo kancelářském prostředí využívána i technologie ZigBee. Ta není energeticky náročná a tím pádem je vhodnější pro bateriově napájené aktuátory a senzory. ZigBee navíc podporuje také stromovou a síťovou (mesh) topologii, kdežto Wi-Fi pouze základní hvězdicovou topologii. Wi-Fi i ZigBee spadají do sítí kategorie PAN, případně WLAN nebo LAN. Jedná se o sítě s krátkým dosahem, typicky desítky metrů. [7]

Zařízení s dosahem na větší vzdálenosti, které jsou součástí sítí MAN nebo WAN (stovky metrů až jednotky kilometrů) se připojují do sítě pomocí jiných technologií. V případě, že se jedná o energeticky nezávislé zařízení s vyšší datovou propustností (desítky kB za den) nebo požadavkem na kontinuální přenos dat, je na místě použití mobilní sítě GSM a jimi podporovaných technologií 3G, LTE nebo 5G. Typickým zástupcem takových zařízeni jsou například IP kamery. [7]

Druhou skupinu zařízení s větším dosahem pak tvoří zejména ta energeticky závislá (bateriově napájená). Tato zařízení pak periodicky zasílají data, která se pohybují maximálně v desítkách bajtů. Jedná se o data bez časového kontextu, která jsou až po přijetí opatřena časovým razítkem. Zařízení komunikují až na výjimky jednosměrně a díky periodickému zasílání a jednosměrné komunikaci nemusí byt trvale přihlášena do sítě, což výrazně snižuje spotřebu elektrické energie a zvyšuje výdrž celého zařízení. Území České republiky je pokryto sítěmi **Sigfox**, **LoRaWAN** a **NarrowBand IoT**, které splňují výše uvedené definice. Technologie a přístupové body sdílí s GSM infrastrukturou současných mobilních operátorů. [7]

Opačnou skupinu, a to krátko-dosahových zařízení tvoří ta, která komunikují pomocí Bluetooth technologie. Jedná se zejména o personální elektroniku, jako jsou chytré náramky, lokátory, zámky, ...atd. Tato zařízení nejsou přímou součástí nějaké velké sofistikované sítě a připojují se přímo s obsluhovaným zařízení jako je například mobilní telefon. [8]

1.4 Testbed a jeho využití

V souvislosti se čtvrtou průmyslovou revolucí a využitím chytrých továren je třeba vytvořit prostor, který by sloužil pro testování a ověřování nových technologií, které mají být uvedeny do provozu. Pro takový výše popsaný prostor se ustálil pojem testbed. [9]

Jako testbed se označuje platforma pro opakovatelné testování nových technologií. Je to určitá množina strojů a zařízení, která umožňuje jednoduše sestavit experimentální výrobní linku. Na této lince jsou posléze testovány principy komunikace a interoperability mezi jednotlivými částmi testbedu. Cílem je zjistit, jestli je navržená konfigurace vhodná k přímému použití ve výrobě, nebo zda-li je možné některé části linky vylepšit k vetší optimalizaci budoucího výrobního procesu. Součástí testbedu je krom dopravníkových pásů a robotických ramen, která slouží pro manipulaci s výrobky, také zařízení sloužící pro identifikaci, optickou inspekci a automatické testování výrobků. [10]

Využití testbedu přináší celou řadu výhod. Po odzkoušení je možné hotovou, sestavenou a optimalizovanou linku přenést do výroby, nebo ji rozebrat a znovu sestavit k otestování dalšího výrobního procesu. Celý tento proces se může také odehrát ve virtuálním prostředí. Tato cesta přináší větší úsporu času i finančních prostředků pro realizaci. V tomto případě se pak již hovoří o digitálním dvojčeti. [9]

Kapitola 2

Tvorba aplikací pro mobilní platformy

Mobilními zařízeními nejsou pouze mobilní telefony. V posledních letech se k ním řadí i chytré hodinky, chytrá auta nebo chytré televize (Smart TV), které mohou ovládat další chytré spotřebiče v domácnosti. Na všech těchto zařízeních je možné spouštět různé aplikace a uzpůsobovat si tak dané zařízení ke své potřebě. Aplikace, které si uživatel na dané zařízení nainstaluje jsou aplikační nástavbou a běží na operačním systému daného zařízení. Ten jim poskytuje tak jako každý jiný operační systém potřebnou paměť a hardwarové prostředky.

2.1 Přehled mobilních operačních systémů

Na trhu jsou dostupné různé operační systémy pro mobilní zařízení. Každý systém má své odlišnosti a svá specifika. V této kapitole se podíváme na nejběžněji používané [11] a historicky významné operační systémy současnosti, a to **Android**, **iOS** a **Windows Mobile**.

2.1.1 Android

Nejrozšířenějším operačním systém je Android od společnosti Google. Toto prvenství si drží i díky své otevřenosti. Systém je možné libovolně upravovat jak po funkční, tak po grafické stránce. Výrobci mobilních zařízení (Samsung, Xiaomi, Huawei, ...atd.), kteří tento operační systém využívají si jej graficky přizpůsobují a dovybavují různými aplikacemi třetích stran.

Operační systém Android je úzce provázán s jeho výrobcem. Služby Google účtu jako Gmail, Google Drive nebo Google Maps jsou integrovány do systému. Uživatel tak po přihlášení má synchronizovaná svá data, zprávy, fotky, polohu a plno dalších nastavení. Zároveň je možné celé zařízení pomocí Googlu lokalizovat, nebo obnovit ze zálohy. Uživatelé si pomocí Google účtu mohou svá zařízení spravovat a instalovat si do něj další aplikace z internetového obchodu Google Play. Ten zároveň kontroluje obsah aplikací proti nevhodnému obsahu nebo přítomnosti virů. Android díky své otevřenosti a mnohoúčelnosti není tak výkonný a optimalizovaný pro daný hardware jako například konkurenční produkt iOS. [12]

Aplikace pro operační systém Android se primárně vyvíjí v objektově orientovaném programovacím jazyce Java nebo v jednodušším a netypovém jazyce Kotlin. Implementace je možná i jazyce C a jeho derivátech. Dominantním vývojovým prostředkem je Android Studio, který umožňuje přímý vývoj, kompilaci a ladění na koncovém fyzickém nebo emulovaném zařízení. Z finančního hlediska není žádný klíčový software zpoplatněn. [13]

2.1.2 iOS

Protipólem operačního systému Android je konkurenční a druhý nejpoužívanější systém pro mobilní platformy – iOS. Jedná se o produkt americké společnosti Apple, která je známa svým novátorstvím v oblasti osobní výpočetní techniky. Oproti předchozímu popsanému systému není otevřený a nelze jej libovolně upravovat. Je distribuován výhradně v původním hardwaru firmy Apple. Díky tomu je systém výkonnější a provoz zařízení je spolehlivější. [14]

Uživatelé mají k dipozici obdobně jako u systému Android jeden účet. Ten umožňuje rovněž synchronizaci dat a přenos mezi jednotlivými zařízeními. Pro instalaci dalších aplikací se využívá Apple Store, kde se uživatelům nabízí celá řada aplikací. Jiná varianta instalace aplikací neexistuje. Pro odemčení systému je nutný tzv. jail break. Poté je systém otevřen instalaci aplikací mimo oficiální distribuci a případným úpravám. Zařízení takto ale ztrácí tovární záruku. [14]

Aplikace na tento operační systém se vyvíjí v objektově orientovaném programovacím jazyce Swift a Objective-C, které jsou produkty firmy Apple. Celý vývoj a následné testování probíhá ve vývojovém prostředí Xcode. V dnešní době je potřebné mít mobilní aplikace dostupné pro obě nejpoužívanější platformy, a proto se čím dále častěji vývoj mobilních aplikací přesouvá na technologie jako je React Native, či Xamarin, které umožňují multiplatformní vývoj pro více operačních systémů současně. [15]

2.1.3 Windows 10 Mobile

Jedná se o alternativní platformu vůči předchozím dvou zmíněným. Výrobce, firma Microsoft, měla za cíl sjednotit vzhled a ovládání s desktopovou verzí. Uživatelé mohli aplikace rovněž stahovat z oficiálního obchodu, Windows Store, který je shodný s obchodem pro desktopovou verzi Windows. [16]

V prosinci roku 2019 Microsoft oficiálně ukončil podporu této mobilní platformy. Nadále tak nevydává aktualizace, neposkytuje podporu a výrobci již neuvádějí na trh nová zařízení. [16]

Aplikace pro Windows 10 Mobile jsou vyvíjeny ve frameworu .NET v jazyce C# podobně jako formulářové aplikace pro standardní desktopové Windows systémy. [17]

2.2 Struktura mobilní aplikace v operačním systému Android

V nativním operačním systému Android může být v jeden okamžik viditelná pouze jedna aplikace, avšak vykonávat nějakou práci muže v jeden okamžik aplikací více. Operační systém přiděluje jednotlivým aplikacím prostředky a v případě potřeby je může také na úkor jiných aplikací skrývat (příchozí hovor, budík) nebo ukončovat (nedostatek paměťových prostředků, ANR). [18]

2.2.1 Aktivita

Aktivita je základním stavebním kamenem každé aplikace. Skládá se z front-endové části (view), která obsahuje jednotlivé grafické prvky aplikace a back-endové části, která reaguje na uživatelské podněty a na jejich základě vykonává předem danou posloupnost instrukcí. Aktivita může získávat data ze sítě nebo z vestavěných senzorů, měnit obsah view, ukončovat nebo spouštět nové aktivity a předávat jim data.

Aktivita se řídí svým životním cyklem. Poté co je aktivita spuštěna operačním systémem nebo jinou aktivitou je volána metoda `onCreate()`. V této metodě se načítá view a všechny jeho prvky se inicializují. Dále se získávají data, která byla předána z předchozí aktivity, nebo jsou uložena z předchozího spuštění aplikace. V této metodě se také registrují callbacky (zpětná volání) pro jednotlivé prvky view, jejichž program se vykoná po jejich aktivaci (tlačítka, výběrové položky, checkboxy, atd.). [19]

Po vytvoření aktivity se volá metoda `onStart()`. Tato metoda nemá příliš velký význam ihned po vytvoření aktivity, ale až při jejím znovu spuštění. Tedy v případě, že byla aktivita předtím přerušena a nyní se do ní uživatel navrátil. V ten moment se v této metodě načtou uložené stavy získané v okamžik, kdy byla aktivita jinou aktivitou nebo systémem přerušena. Uložené stavy nesou například informace o obsahu jednotlivých grafických prvků, které je třeba před znovu spuštěním aplikace obnovit. V této metodě je aktivita také zviditelněna uživateli. [19]

Těsně před spuštěním je systémem volána metoda `onResume()`. V této metodě se mohou navazovat spojení, aktivovat přístup k získaným hardwarovým prostředkům nebo spouštění dalších vláken. Po vykonání veškerých instrukcí je aktivita připravena na uživatelskou interakci. [19]

V případě, že je aktivita překryta jinou aktivitou, ale nadále je viditelná, je operačním systémem nejprve volána metoda `onPause()`. Aktivita je v ten okamžik pozastavena a není již v popředí. Aktivita může být avšak nadále viditelná, pokud uživatel pracuje v režimu více oken. V rámci této metody by mělo dojít k uvolnění prostředků. Tím může být přístup k fotoaparátu, GPS, nebo ostatním senzorům, ke kterým aktivita již nepotřebuje přístup. Pokud je uživatel vrácen zpět do aktivity, je opět volána metoda `onResume()` a dříve uvolněné prostředky jsou znovu získány tak,aby byla zachována celistvost dané aktivity. [19]

Ne vždy se ale uživatel okamžitě vrací zpět do aktivity. Pokud operační systém potřebuje vyvolat na popředí jinou aktivitu, nebo uživatel aplikaci vědomě skryje, je volána metoda `onStop()`.

V této metodě by měl programátor zajistit uložení obsahu grafických prvků a ukončit veškerou práci aktivity, která může nadměrně zatěžovat CPU. V případě, že se uživatel do aktivity vrátí, je volána metoda `onRestart()` a následně metoda `onStart()`, kde se uložené stavy opět načtou a aktivita pokračuje. [19]

Poslední metoda životního cyklu aktivity se nazývá `onDestroy()`. Jedná se o poslední možnost, kde by měly být uvolněny všechny prostředky, které nebyly uvolněny v předchozích dvou popsaných metodách. Je třeba zmínit, že volání této metody nemusí proběhnout vždy. Pokud operační systém potřebuje uvolnit nutně paměť, tak v případě násilného ukončení procesu nemusí proběhnout celý životní cyklus. Z toho vyplývá, že by v této metodě neměly být žádné kritické sekce kódu, jako je například uložení souborů nebo zavření databázového spojení. [19]

Životní cyklus obecné aktivity včetně posloupnosti volání jednotlivých metod je znázorněn pomocí diagramu aktivit na obrázku 2.1.

2.2.2 Služba

Druhou možností jak nechat aplikaci vykonávat nějakou činnost je služba. Služba oproti aktivitě nemá front-endovou část, ale pouze část back-endovou. Její běh je nezávislý na běhu a životnosti aktivit. V operačním systému rozlišujeme celkem tři druhy služeb. [20]

První je služba, která běží v popředí. Vykonává tak operace, které jsou pro uživatele vnímatelné. Může se jednat například o přehrávání hudby.

Druhým typem služby je ta, která běží na pozadí. Taková služba může například hlídat příchozí hovory nebo nové zprávy.

Třetím a posledním typem služby je služba vázaná. Vázaná služba běží pouze tak dlouho, dokud jsou k ní vázány jiné komponenty aplikace (vetšinou aktivity). Jakmile služba ztratí veškeré vazby na tyto komponenty, sama se zničí. Vázaná služba může například získávat pravidelně polohu zařízení, a to i v případě, že je aktivita aplikace, která polohu následně vizualizuje skrytá. V případě zničení aktivity, zanikne i služba, která se s danou aktivitou váže, protože další určování polohy již nemá smysl.

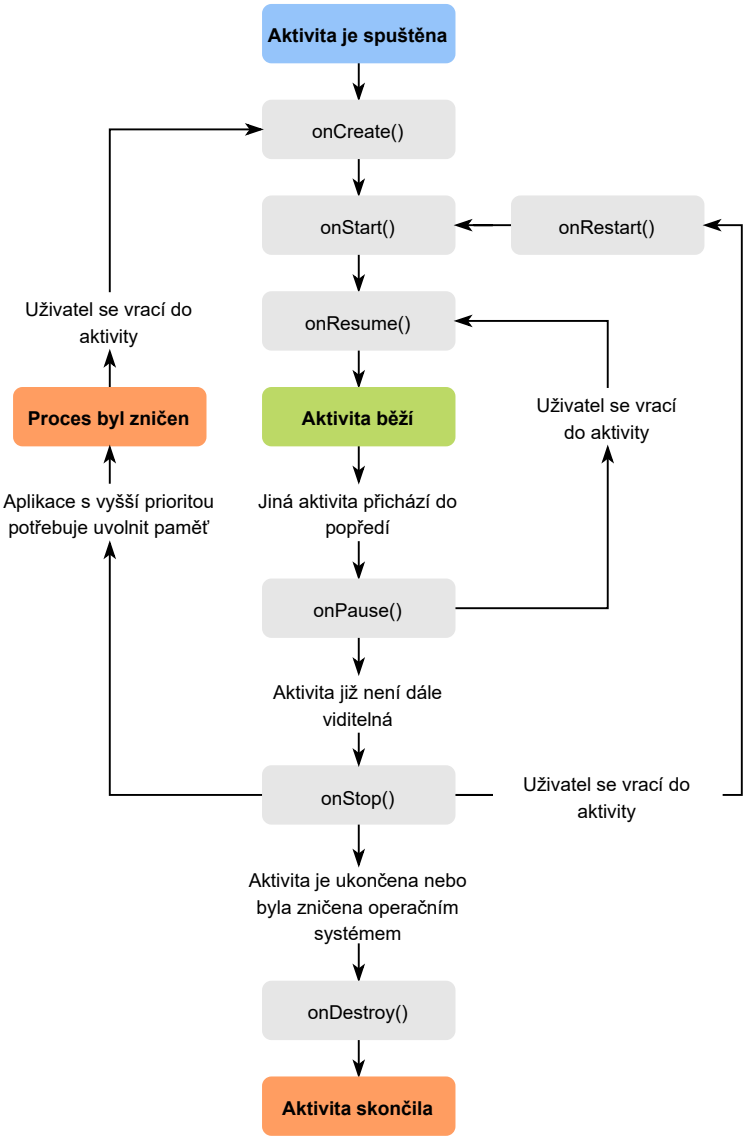
Životní cyklus služby je jednodušší než životní cyklus aktivity. Služba může být spuštěna dvěma způsoby.

První způsob je spuštění aktivitou pomocí zavolání metody `startService()`. Ve službě slouží pro odchycení jejího spuštění callback s názvem `onStartCommand()`, kterému předchází ještě jedna, operačním systémem volaná metoda `onCreate()`. Po této sekvenci je služba spuštěna a připravena k používání. Zastavení služby se provádí voláním metody `stopService()` z aktivity (nemusí být ze stejné aktivity, která službu spustila), případně voláním metody `stopSelf()` z běžící služby, která se tak může sama zastavit. Poslední systémem volaná metoda je pak `onDestroy()`, po níž je instance služby definitivně zničena. [20]

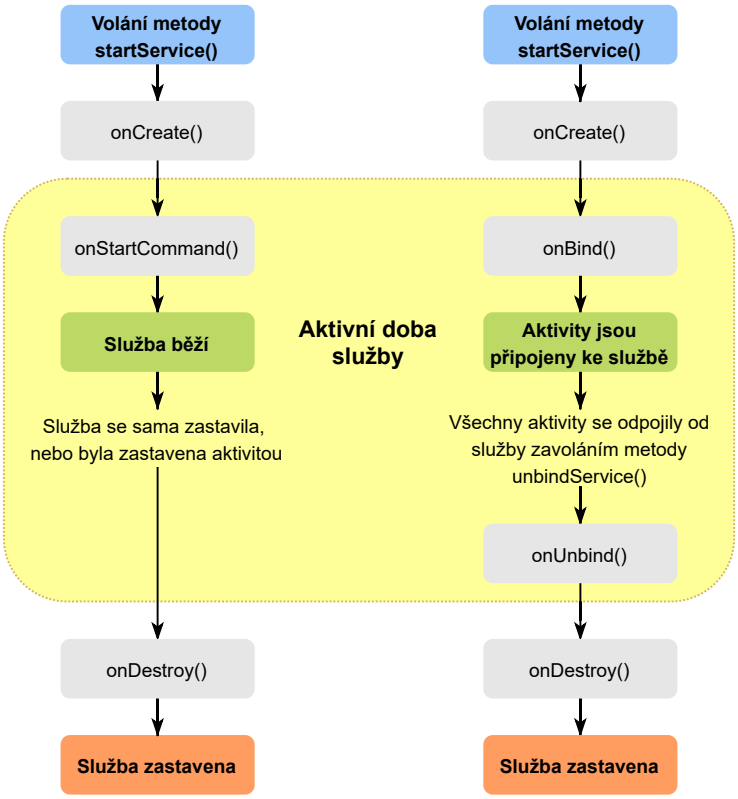
Druhý způsob spuštění služby a její svázání s aktivitou je pomocí metody `bindService()`. Po spuštění služby je volán návazný callback `onBind()`, kterému předchází systémem volaná metoda `onCreate()`. Po této sekvenci služba běží jako vázaná a k jejímu zastavení je nutné přerušení vazby s aktivitou. To se provádí voláním metody `unbindService()`, která volá návazný callback ve službě – `onUnbind()`. Poslední systémem volaná metoda je pak opět `onDestroy()`. Pouze vázaná služba může s aktivitou sdílet paměťový prostor a vyměňovat si data. [20]

Na následujícím vývojovém diagramu je možné vidět životní cyklus obecné služby. Z diagramu je patrné, že je možné například vytvořit vazbu k již spuštěné službě. Toho lze například využít v případě, že k dané službě přistupuje v rámci návrhu aplikace více aktivit a zároveň nechceme během přechodu mezi nimi službu zničit, a tím pádem například ztratit spojení s jiným zařízením (Bluetooth, externí databáze). [20]

Životní cyklus obecné služby včetně posloupnosti volání jednotlivých metod je znázorněn pomocí diagramu aktivit na obrázku 2.2.



Obrázek 2.1: Životní cyklus aktivity, zdroj: autor



Obrázek 2.2: Životní cyklus služby, zdroj: autor

Kapitola 3

Bluetooth Low Energy

Bluetooth je otevřený bezdrátový komunikační standard, který vynalezla firma **Ericsson** v roce 1996 jako náhradu za standardní sériové rozhraní RS-232. Název dostal po přídomku dánského a norského krále, který nesl jméno Harald I. „Modrozub" Gromsson. Bluetooth technologie je definovaná standardem IEEE 802.15.1, dle kterého spadá do kategorie sítí typu PAN. [21]

Od uvedení první verze v roce 1996 vzniklo již více než pět verzí této technologie. Bluetooth pracuje v ISM pásmu 2,4 GHz podobně jako například technologie Wi-Fi. Ke kódování zasílaných dat je využito metody FHSS, která umožňuje existenci více systémů (teoreticky až 26) v jednom prostředí ve stejném čase. Díky tomu je možné využívat více zařízení Bluetooth tak, aniž by se navzájem rušily.

V dnešní době je technologií Bluetooth vybavena téměř veškerá personální elektronika. Ke komunikaci ji používají mobilní telefony, chytré hodinky, klávesnice, myši, sluchátka nebo lokátory.

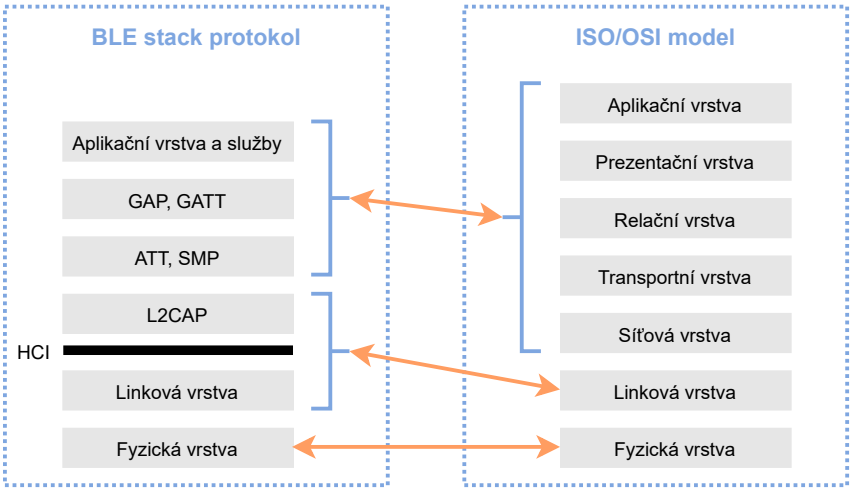
3.1 Hlavní vlastnosti a využití technologie Bluetooth Low Energy

Specifikace Bluetooth 4.0 (Low Energy) byla vydána v červenci 2010. Jednalo se o první specifikaci, která byla energeticky nenáročná a předem určená pro zařízení s nízkými datovými toky. Tato konkrétní specifikace tedy není vhodná ke streamování, ale přenosu krátkých (několika bajtových) zpráv oběma směry. [22]

Bluetooth 4.0 je tak vhodné například k přenosu naměřených biologických veličin z chytrých hodinek do mobilního telefonu. Další oblastí použití může být orientace a navigace uvnitř budov, kde klasické lokalizační systémy jako GPS nebo Galileo nefungují. Uživatel se pomocí mobilní aplikace v chytrém telefonu neustále dotazuje na staticky rozmístěné Bluetooth zařízení uvnitř budovy. Na základě jejich ID, síly signálu a vysílacím výkonu, který aplikace průběžně zjišťuje, je vypočítána poloha uživatele a zobrazena na mapě.

3.2 Vrstvy protokolu Bluetooth Low Energy

Obdobně jako ostatní formy komunikace v počítačových sítí, které je možné popsat pomocí ISO/OSI modelu, lze komunikaci pomocí Bluetooth a také BLE popsat pomocí Bluetooth stack protokolu, který je na referenčním ISO/OSI modelu založen. Bluetooth stack protokol se skládá ze dvou hlavních částí – Controller a Host. [23]



Obrázek 3.1: Srovnání BLE stack protokolu s referenčním ISO/OSI modelem, zdroj: autor

- **Controller** zahrnuje fyzickou a linkovou vrstru a také rozhraní HCI pro komunikaci s vyššími vrstvami.
  - **Fyzická vrstva** je nejnižší vrstvou celého stacku. Zajišťuje komunikaci pomocí rádia na vyčleněných kanálech a přímo předává data jinému Bluetooth zařízení.
  - **Linková vrstva** je druhou a poslední vrstvou komunikační části Bluetooth stack protokolu. Tato vrstva zahajuje a ukončuje spojení, ovlivňuje strukturu paketů, vybírá komunikační kanály a zajišťuje bezpečnost komunikace.
- **Host** zahrnuje ostatní vyšší vrstvy protokolu, kde se pracuje s daty na vyšší úrovni. Tyto vrstvy jsou implementovány jako součásti operačního systému oproti komunikační části, kde se jedná fyzické součásti (čipy, antény, atd.). Vyšší vrstvy komunikují s linkovou a fyzickou vrstvou opět pomocí rozhraní HCI.
  - **Logical Link and Adaptation Protocol** je někdy označován zkratkou L2CAP. Tato vrstva má za úkol předávat pakety na rozhraní HCI. V referenčním ISO/OSI modelu bychom ji zařadili ještě do linkové vrstvy.
  - **Attribute Protocol** přenáší atributy mezi klientem a serverem, které jsou definovány v GATT. Attribute Protocol také příchozí data rozděluje do daných atributů jako je typ,

hodnota, oprávnění, .atd. Security Manager se stará o párování, ověřování a šifrování dat mezi zařízeními.

– **GAP** a **GATT** definují, jestli je dané zařízení serverem nebo klientem, a jakou má roli během vytváření spojení.

– **Aplikační vrstva a služby** jsou již koncové aplikace, které běží v rámci GATT protokolu. Existuje již několik předdefinovaných profilů. Například pro správu baterie zařízení (BAP), měření srdečního tepu (HRP), nebo lokalizace (LNP). Zároveň je možné vytvořit vlastní profily s vlastním komunikačním rozhraním.

3.3 GAP – Generic Access Profile

Generic Access Profile je důležitou a neoddělitelnou součástí technologie Bluetooth Low Energy. GAP specifikuje, jak se mají zařízení ostatních výrobců chovat při objevování a připojování, vysílání a přijímání dat a také vytváření zabezpečeného spojení. BLE zařízení může v jeden okamžik pracovat v jednom nebo více režimů GAP.

Vysílač/Přijímač (Broadcaster/Observer) je režim, kdy zařízení jednosměrně, všesměrově a bez vytváření spojení vysílá data.

- **Vysílač (Broadcaster)** periodicky vysílá pakety dat bez potvrzení, že byly veškeré pakety doručeny a žádné se cestou neztratily.
- **Přijímač (Observer)** přijímá data, která vysílače vysílají.

V této topologii může být více vysílačů i přijímačů zároveň. Jako příklad můžeme uvést systém, kde jeden nebo více mobilních telefonů (přijímačů) vyhledává a přijímá data z více vysílačů, kterými mohou být předem rozmístěné senzory v daném prostoru. Tento režim je značně podobný protokolu UDP. Ten rovněž podporuje všesměrové vysílání bez validace doručených dat nebo ztráty jednotlivých paketů.

Druhým typem režimu je **Peripheral/Central** režim, ve kterém probíhá obousměrná komunikace mezi zařízeními, která navázala spojení.

- **Peripheral** vystupuje jako zařízení typu slave, které vysílá pakety umožnující centrálním zařízením je nalézt. Po připojení a vytvoření spojení přestávají pakety zasílat, což umožňuje takové zařízení konstruovat jako nízko-spotřební.
- **Central** vyhledává a zahajuje vytváření spojení s peripheral zařízením. Vystupuje tedy jako master. Zařízení typu central může být připojeno k více zařízením typu peripheral zároveň.

Tento režim je vhodný pro komunikaci v systémech, kde je jeden centrální uzel (master) a více podřízených zařízení (slave). Master dle potřeby naváže spojení s daným slave a vyměňuje si s ním

data. Rozdílem oproti předchozímu popsanému režimu je obousměrná komunikace a exkluzivní přístup k zařízení. [24]

3.4 GATT – Generic Attribute Profile

Oproti Generic Access Profile, který definoval, jak zařízení komunikují, Generic Attribute Profile definuje, jakým způsobem si zařízení budou mezi sebou vyměňovat data. Protokol, který GATT definuje, je založen na konceptu služeb (services) a vlastností (characteristics). Všechny tyto informace jsou uloženy ve vyhledávací tabulce.

GATT je možné použít pouze tehdy, je-li mezi zařízeními ustanoveno spojení. Zařízení tedy musela projít procesem navázání spojení v režimu Peripheral/Central. V režimu Broadcaster/Observer nelze GATT použít. Jelikož existuje spojení, je přístup k zařízení s roli Peripheral exkluzivní a může být k němu v jeden okamžik připojeno pouze jedno zařízení v roli Central. Jakmile je zařízení Peripheral připojeno, není již dále viditelné pro ostatní zařízení a dokud spojení neskončí, není možné, aby jej vyhledalo, nebo se k němu připojilo jiné zařízení.

Mezi zařízeními s navázaným spojením funguje vztah klient – server. Server obsahuje definice služeb (services) a vlastností (characteristics), do kterých vystavuje data. Klient se pak serveru dotazuje a data z jednotlivých služeb a vlastností získává. Všechny transakce spouští klient, který následně přijímá odpovědi od serveru. Klientem nemusí být vždy zařízení s rolí Central a serverem Peripheral zařízení. Role jednotlivým zařízením přiděluje GAP. I když to není obvyklé, tak Peripheral zařízení může vystupovat jako klinet a Central jako server. [25]

3.4.1 Profily, služby a vlastnosti

Přenos dat pomocí GATT je založen na generickém datovém typu, který je nazýván profil.

Profil je souhrnné označení pro kolekci služeb a vlastností. Tyto kolekce jsou předefinované standardizační organizací Bluetooth Special Interest Group (SIG) a zajišťují komunikační kompatibilitu napříč zařízeními a aplikacemi. Pokud výrobce zařízení pro měření srdečního tepu implementuje profil dle standardu Bluetooth SIG, je pravděpodobné, že většina aplikací, které rovněž implementují tento profil, bude schopno data ze zařízení vyčíst, aniž by to bylo explicitně ověřeno. Jejich kompletní a pravidelný seznam je možné nalézt na webu Bluetooth SIG. Je možné provést vlastní definici profilu za účelem vytvoření vlastní komunikační strategie.

Služba (service) dělí data v profilu na logické celky. Obsahuje objekty zvané vlastnosti (characteristic), které jsou koncovým nositelem dat. Každá služba musí mít minimálně jednu vlastnost. Může jich být ale definováno i více. Jedinečným identifikátorem každé služby je až 128bitové číslo, které se nazývá UUID. Některé služby jsou v rámci profilu mandatorní a není možné je vynechat.

Nejnižší úrovní GATT je pak vlastnost. Vlastnost je nositelem jedné datové hodnoty, která je reprezentována posloupností bajtů. Stejně jako služba je i vlastnost identifikovatelná pomocí

UUID. Koncová zařízení si svá data vyměňuji přímo pomocí vlastností. Klient může ze serveru vlastnost číst, zapisovat do ní, nebo si nechat její obsah asynchronně zasílat (notifikovat), jestliže dojde ke změně. Kromě UUID a dat obsahuje vlastnost také atributy, které k ní modifikují přístup. Vlastnost může být určená pro čtení, zápis, notifikace, nebo kombinace zmíněných atributů. Součástí vlastnosti jsou také deskriptory. Ty mohou uchovávat informace o názvu vlastnosti, případně jejím popisu nebo o stavu zasílání notifikací (povoleno/nepovoleno). [25]

Díky své generičnosti je GATT jednoduše implementovatelný v objektově orientovaných programovacích jazycích.

Kapitola 4

Popis loggeru pro testbed

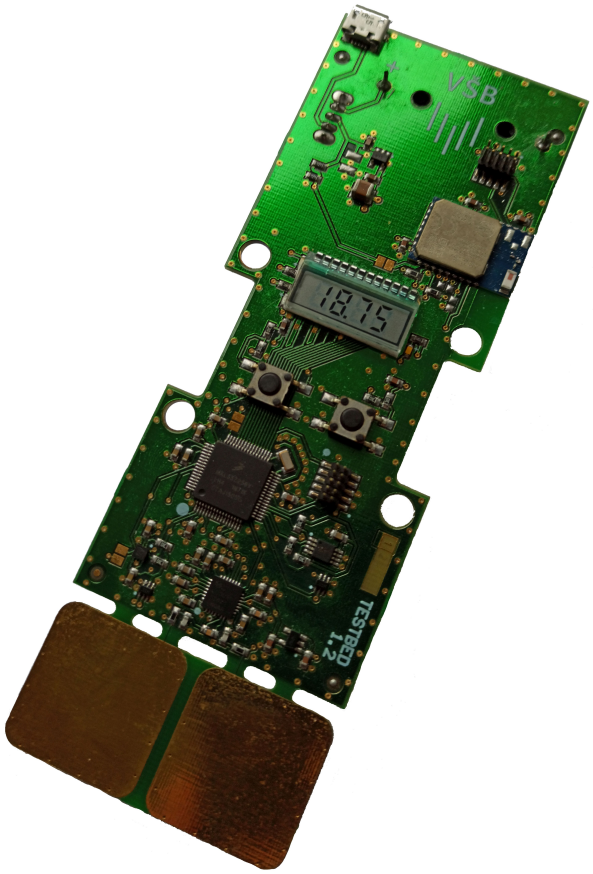
Pro účely nově budovaného testbedu na Vysoké škole báňské – Technické univerzitě Ostrava bylo navrženo a vyvinuto jednoduché zařízení, které umí komunikovat se senzory na nejnižší úrovni, zpracovat naměřená data a odeslat je skrze technologii Bluetooth Low Energy. Zařízení je součástí diplomové práce Bc. Daniela Kajzara, který je rovněž jeho autorem.

Základem zařízení je čtyřvrstvá deska plošných spojů o rozměrech 134 × 46 mm. Vnější povrchy desky jsou chráněny zelenou nepájivou maskou a opatřeny bílým servisním potiskem. Povrch pájecích a kontaktních plošek je upraven pomocí imerzního zlata (ENIG). DPS byla strojově osazena a posléze přetavena v pájecí peci. Produkt testbed je vyobrazen na obrázku 4.1.

Srdcem celého zařízení je mikrokontrolér **MK33Z256V**. Pomocí sériových rozhraní komunikuje s krokoměrem (**KX126-1063**), senzorem srdečního tepu (**MAX30003**) a teplotním senzorem (**AT30TS74-XM8M-T**). Naměřené hodnoty zařízení zobrazuje na čtyřmístném sedmisegmentovém LCD displeji a odesílá pomocí Bluetooth Low Energy modulu (**BLE112**). K ovládání zařízení se používá dvojice mikrotlačítek. Zařízení je napájeno z 3,6 V lithiové baterie typu 1/2AA, nebo pomocí micro USB konektoru.

Levým tlačítkem se v kruhu přepíná mezi zobrazením jednotlivých údajů. Pravým tlačítkem se zobrazení údajů na displeji vyvolá. Zobrazit lze celkový úhrn kroků, poslední změřenou hodnotu srdečního tepu a teploty a také jedinečné 17bitové ID zařízení, které je reprezentované v hexadecimální tvaru a podle nějž lze zařízení identifikovat, včetně aktuálního osazení senzorů. Kroky se na displeji automaticky aktualizují a odesílají do Bluetooth modulu po zaznamenání 12 kroků. Teplota se měří automaticky každých 30 sekund a v případě změny oproti předchozí teplotě je hodnota aktualizována. Měření srdečního tepu je nutné iniciovat stiskem pravého tlačítka. Samotné měření je podmíněno správným uchopením měřících elektrod uživatelem a zachycením srdečního rytmu. Po provedení měření je hodnota aktualizována.

Zařízení nemá EEPROM paměť a po odpojení napájení jsou poslední údaje ztraceny. Zařízení rovněž nedisponuje obvodem reálného času a tím pádem nemají pořízená data časový kontext.



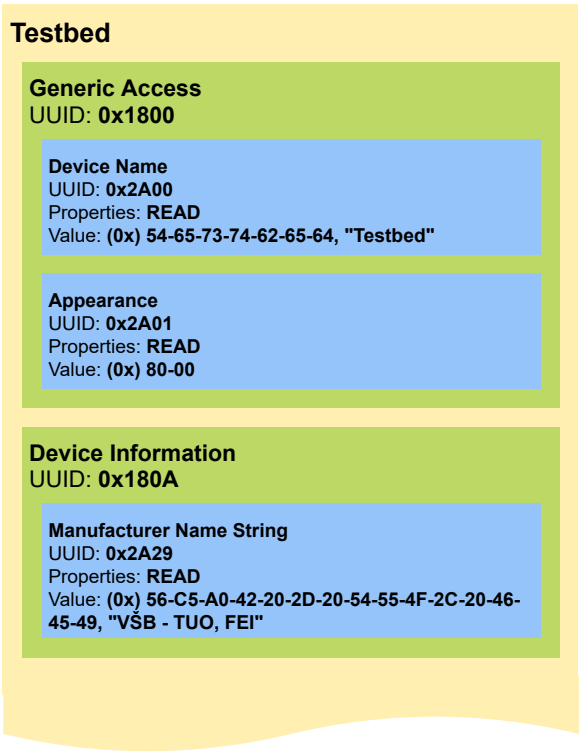
Obrázek 4.1: Produkt testbed, zdroj: autor

Zařízení může být osazeno ve variantách (plné nebo částečné osazení senzorů). V té souvislosti je možné při neosazení senzoru srdečního tepu DPS zkrátit o měřicí elektrody.

4.1 GATT profil

Zařízení komunikuje s okolím pomocí Bluetooth Low Energy modulu. Pro komunikaci byl navržen vlastní komunikační profil s jednou službou, která obsahuje následující čtyři vlastnosti: ID zařízení, celkový úhrn kroků, poslední hodnotu srdečního tepu a teploty.

Kromě vlastní služby obsahuje profil také další dvě, které nesou údaje o názvu zařízení a jeho výrobci. První služba s hodnotou UUID `0x1800` má název Generic Access. Jedná se o povinnou službu každého profilu, která je standardizovaná organizací Bluetooth SIG. Proto je její UUID kratší než u služeb, které jsou vlastního návrhu. Služba obsahuje dvě vlastnosti. První s UUID `0x2A00` obsahuje textový řetězec s názvem zařízení tak, jak se zobrazí při vyhledání. V případě loggeru má vlastnost atribut pouze pro čtení a obsahuje konstantní hodnotu `Testbed`. Druhá vlastnost



Obrázek 4.2: Struktura GATT profilu produktu testbed – 1. část, zdroj: autor

s UUID `0x2A01`, a opět s atributem pouze pro čtení, obsahuje informaci o samotném zařízení. V případě loggeru se jedná o pole s konstantní hodnotou. `{0x80, 0x00}`

Druhá služba s UUID `0x180A` je opět povinná a standardizována organizací Bluetooth SIG. Má pouze jednu vlastnost s atributem pouze pro čtení (UUID `0x2A29`) a obsahuje konstantní textový řetězec s údajem o výrobci zařízení. Její hodnota je `VŠB-TUO, FEI`. Strukturu mandatorních služeb můžeme vidět na obrázku 4.2.

Služba, která obsahuje naměřené hodnoty ze zařízení není standardizovaná (ani její vlastnosti nejsou standardizované). Proto je její UUID (`21110001-e780-4112-a997-5c3e475adbc3`) delší než u předchozích standardizovaných služeb. Celkem obsahuje čtyři vlastnosti, jež jsou nositelky informací o hodnotách naměřených pomocí senzorů a také ID zařízení.

První vlastnost s UUID `21110005-e780-4112-a997-5c3e475adbc3` a atributem opravňující připojeného klienta pouze ke čtení dat, nese informaci o jedinečném ID zařízení a senzorech, které jsou v dané verzi loggeru osazeny. Data jsou reprezentována jako textový řetězec o délce 5 znaků, který obsahuje pouze číslice od „0” do „9” a písmena „A” až „F”. Číslo na prvním místě řetězce udává informaci o osazených senzorech. Může nabývat pouze hodnoty „0” až „7”. Převedeme-li si číslo na binární reprezentaci tak nejméně významný bit (LSB) indikuje osazení senzoru teploměru, druhý nejméně významný bit pak osazení senzoru srdečního tepu a nejvýznamnější bit (MSB) pak osazení senzoru akcelerometru (počítání kroků). Logická jednička indikuje přítomnost daného senzoru, lo-

Obrázek 4.3: Struktura GATT profilu produktu testbed – 2. část, zdroj: autor

gická nula pak jeho absenci. Je-li například první číslo z tohoto řetězce 4, pak převedené do binární reprezentace jako `0b101` znamená, že je na daném zařízení osazen pouze akcelerometr a teplotní senzor. Senzor srdečního tepu pak chybí. Další a poslední čtyři znaky řetězce pak nesou údaj o ID

zařízení, který je odvozen ze sériového čísla mikrokontroléru. Vlastnost obsahuje jeden deskriptor (UUID `0x2901`), který udává název vlastnosti.

Druhá vlastnost v pořadí s UUID `21110003-e780-4112-a997-5c3e475adbc3` má kromě atributu pro čtení ještě atribut umožnující zasílání asynchronních notifikací. Hodnotou vlastnosti je opět textový řetězec s délkou 3 znaky, který tentokrát obsahuje pouze číslice od „0" do „9". Číselná hodnota, která je z vlastnosti vyčtena, může teoreticky nabývat hodnot `000` až `999` a obsahuje změřenou hodnotu srdečního tepu v pulsech za minutu (BPM). Reálně je však zaražení omezeno na měření hodnot v rozsahu `40` až `180` BPM. Hodnoty mimo tento rozsah jsou za hranicí biologických možností žijícího člověka. Vlastnosti náleží opět deskriptor s UUID `0x2901`, který udává název vlastnosti a navíc ještě jeden deskriptor s UUID `0x2902`, který nese informaci o tom, zda-li klient připojený k serveru povolil zasílání asynchronních notifikací či nikoliv.

Další vlastnost s UUID `21110004-e780-4112-a997-5c3e475adbc3` a stejnými atributy nese informaci o celkovém úhrnu kroků. Jakmile logger změří další načtené kroky, tak hodnotu této vlastnosti o změřenou hodnotu inkrementuje. Hodnota je reprezentována textovým řetězcem o délce 6 znaků, který obsahuje pouze číslice od „0" do „9". Maximální hodnota, kterou zvládne logger odeslat, je 999 999 kroků. V případě, že je hodnota o řád(y) nižší, jsou v řetězci obsažené i vodící nuly. Vlastnosti náleží stejné dva deskriptory jako předchozí popsané.

Poslední vlastností této služby s UUID `21110002-e780-4112-a997-5c3e475adbc3` a stejnými atributy jako vlastnosti pro kroky a srdeční tep, je vlastnost pro poslední změřenou teplotu. Ta je reprezentována textovým řetězcem o délce 7 bajtů v následujícím formátu „+028.50". První znak v řetězci je vždy znaménko, které nabývá hodnot „+" nebo „-". Další tři znaky (včetně vodících nul) nesou údaj o celočíselné hodnotě teploty. Pátým znakem je vždy tečka („."). Poslední dva znaky nesou údaj o desetinné hodnotě teploty (včetně vodících nul). Stejně jako znaky s celočíselným významem mohou nabývat hodnot „0" až „9". Vlastnost také obsahuje stejné dva deskriptory jako předchozí dvě popsané vlastnosti.

Strukturu služeb profilu testbed můžeme vidět na obrázku 4.3.

Kapitola 5

Návrh mobilní aplikace

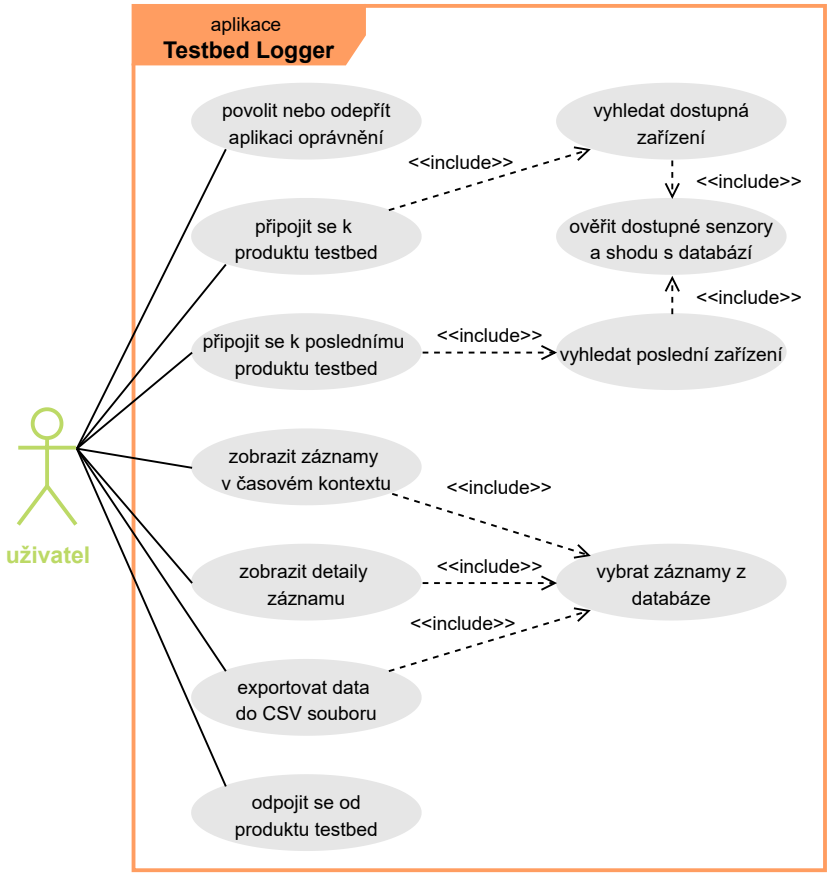
Návrh a implementace uživatelské mobilní aplikace probíhaly podle vodopádového modelu. Nejdříve byly specifikovány požadavky na aplikaci a její komunikaci s deskou testbed. Na základě požadavků pak byl sestaven diagram případů užití.

5.1 Případy užití navrhované aplikace (use case)

Dle specifikace a požadavků na aplikaci se počítá s aplikací, kterou bude (s největší pravděpodobností) využívat na jednom mobilním zařízení pouze jeden uživatel. Veškerá data mají být ukládaná pouze v lokální databázi bez požadavků na synchronizaci. Z toho důvodů figuruje v případu užití pouze jeden typ aktéra, a to běžný uživatel, kterým je zároveň myšlen uživatel mobilního zařízení. Uživatel nemá žádné rodiče ani potomky a může využívat všechny funkcionality, které bude aplikace nabízet.

Jako v každé aplikaci pro operační systém Android může uživatel aplikaci povolit nebo odepřít oprávnění, která daná aplikace vyžaduje pro svůj běh. Požadovaná oprávnění každé aplikace můžeme rozdělit do dvou kategorií. První kategorie obsahuje oprávnění, která nejsou pro běh aplikace existenčně důležité, nicméně zvyšují uživatelský komfort a jejich nepovolení, případné pozdější odepření, může vést ke ztrátě dané funkcionality. V rámci aplikace pro desku testbed patří do této kategorie oprávnění pro používání (externího) úložiště, které slouží pro exportované CSV soubory. Druhou kategorií můžeme nazvat oprávnění, která jsou naprosto nezbytná pro alespoň základní chod aplikace. V tomto konkrétním případě se jedná o oprávnění k poloze a adaptéru Bluetooth. Bez povolení těchto oprávnění nelze zařízení vyhledat a ani se k zařízením připojit. Využití každého oprávnění je vhodné uživateli řádně zdůvodnit a v případě neudělení těch nutných mu nedovolit pokračovat v aplikaci.

Prvním z kroků po spuštění aplikace je připojení k dostupným zařízením. Samotnému procesu připojování předchází a je s ním bezpodmínečně spojeno vyhledaní dostupných zařízení a ověření shody s databází. Proto jsou tyto případy užití vztaženy k ostatním případům vazbou **include**.



Obrázek 5.1: Diagram případů užití v navrhované aplikaci, zdroj: autor

Vyhledáním zařízení lze zjistit pouze jejich název, MAC adresu a sílu signálu (RSSI). Pokud potřebujeme zjistit, jaké je ID zařízení a jakými senzory je daný produkt testbed vybaven, musíme se k desce připojit, zjistit dostupné služby a vyčíst jeho ID a údaj o osazených senzorech. Zároveň se tyto vyčtené údaje porovnávají s databází, aby bylo možné zjistit, zda-li již bylo v minulosti s tímto testbedem navázáno spojení, jestli nebyl pozměněn, nebo jestli je po zapnutí správně inicializován. Teprve poté může uživatel s deskou testbed pracovat.

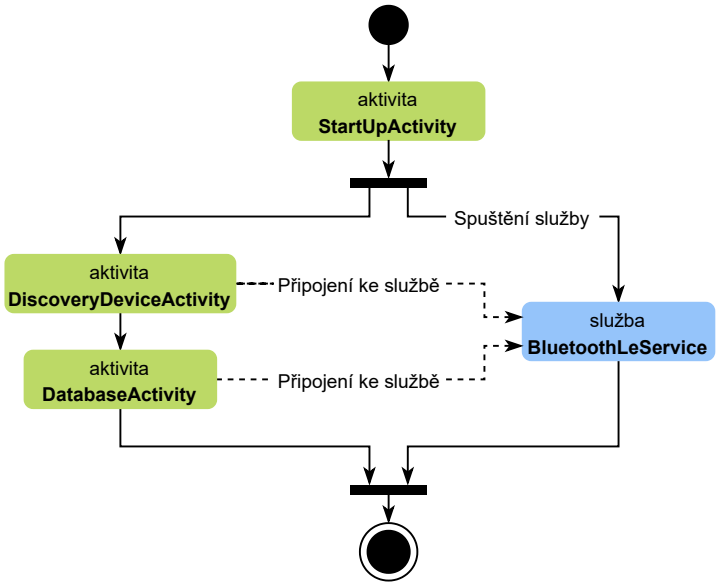
Aplikace uživateli nabízí zobrazení dat v různých podobách. Základním zobrazením je přehled, který vykazuje základní statistické ukazatele (minimální hodnota, medián, maximální hodnota, průměrná hodnota, celkový úhrn) za určité období (poslední den, poslední týden, poslední měsíc). Dalším z nabízených zobrazení, kdy jsou záznamy prezentovány v časovém kontextu, je grafické zobrazení. Uživatel tak může přehledně porovnat dané intervaly za určitý časový úsek (např. dny v měsíci mezi sebou). Poslední z přehledů, které má uživatel k dispozici je přehled jednotlivých záznamů. Ty jsou vyfiltrovány podle období a řazeny sestupně nebo vzestupně podle hodnoty nebo času. U každého záznamu lze navíc zobrazit jeho detaily jako je klíč záznamu, ID záznamu nebo časové razítko.

Mezi nabízené funkcionality aplikace patří také možnost již výše zmíněného exportu uložených zařízení a záznamů do souboru CSV. Je pouze na uživateli a jeho uvážení, které typy dat a ze kterých zařízení nechá exportovat. Tento i předchozí případy užití, které zajišťují prezentaci dat jsou neoddělitelně spojeny s daty uloženými v databázi. Proto jsou tyto případy vázány s případem užití sloužícímu k výběru dat z databáze vazbou **include**.

Posledním z případů užití je možnost odpojit se od právě používaného zařízení a znovu se připojit k jinému produktu testbed. Diagram případů užití je znázorněn na obrázku 5.1

5.2 Návrh struktury aplikace

Návrh struktury aplikace vycházel z předchozí analýzy případů užití. Aplikace se bude skládat ze tří aktivit a jedné služby běžící na pozadí.



Obrázek 5.2: Diagram aktivit zjednodušené struktury navrhované aplikace, zdroj: autor

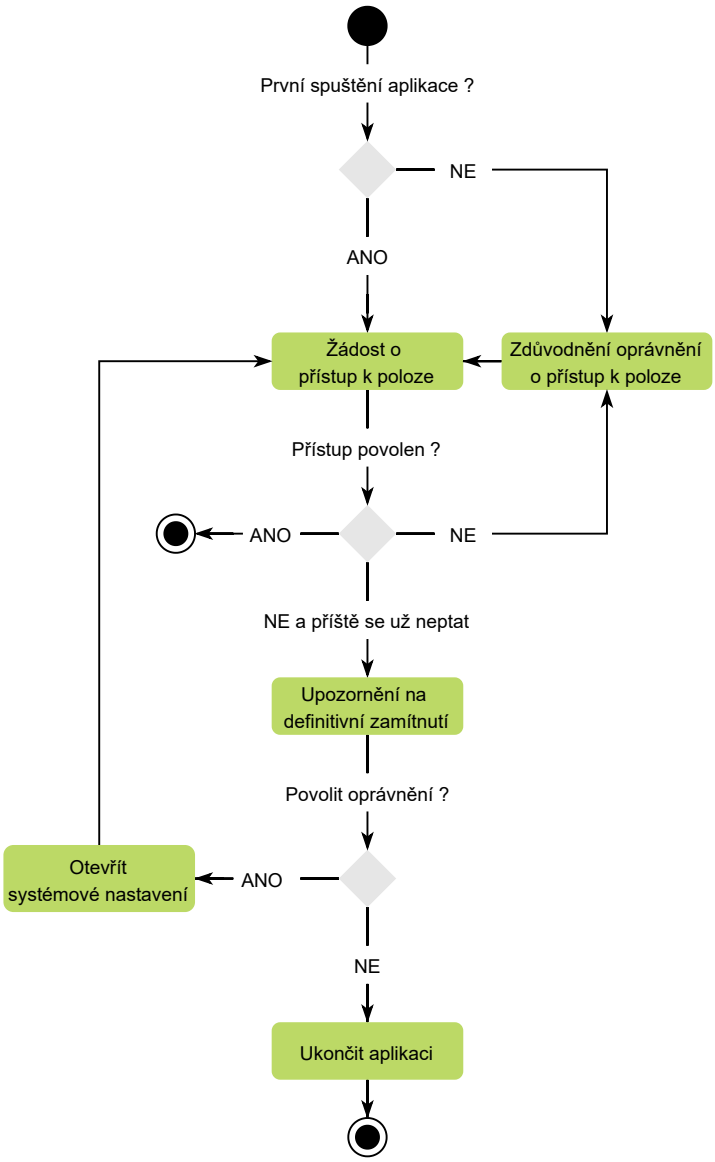
První aktivita provede uživatele přidělením oprávnění, kontrolou zapnutých adaptérů a spuštěním aplikace. Druhá aktivita pak má na starost vyhledání, objevení a připojení se k jednotlivým zařízením. Třetí a poslední aktivita slouží pro prezentaci záznamů, vykreslení grafů a exportu dat do CSV souboru. V pozadí těchto aktivit běží služba, která obsluhuje spojení s produktem testbed, přijímá data a ukládá je do databáze. Zjednodušenou strukturu aplikace můžeme vidět na obrázku 5.2.

5.2.1 StartUpActivity

První aktivitou, která se zavede po spuštění aplikace je aktivita s názvem StartUpActivity. Jak již bylo zmíněno dříve, aktivita má za úkol zkontrolovat stav jednotlivých oprávnění a adaptérů, které

jsou pro chod aplikace nezbytné.

První z podmínek, která je kontrolována je podpora Bluetooth 4.0 LE na zařízení, kde je aplikace spuštěna. Podpora minimálně této verze technologie Bluetooth je nezbytná pro připojení k produktu testbed. Pokud tato podmínka není splněna, je uživatel o této skutečnosti informován a aplikace je posléze ukončena.



Obrázek 5.3: Diagram aktivit povolení oprávnění k poloze, zdroj: autor

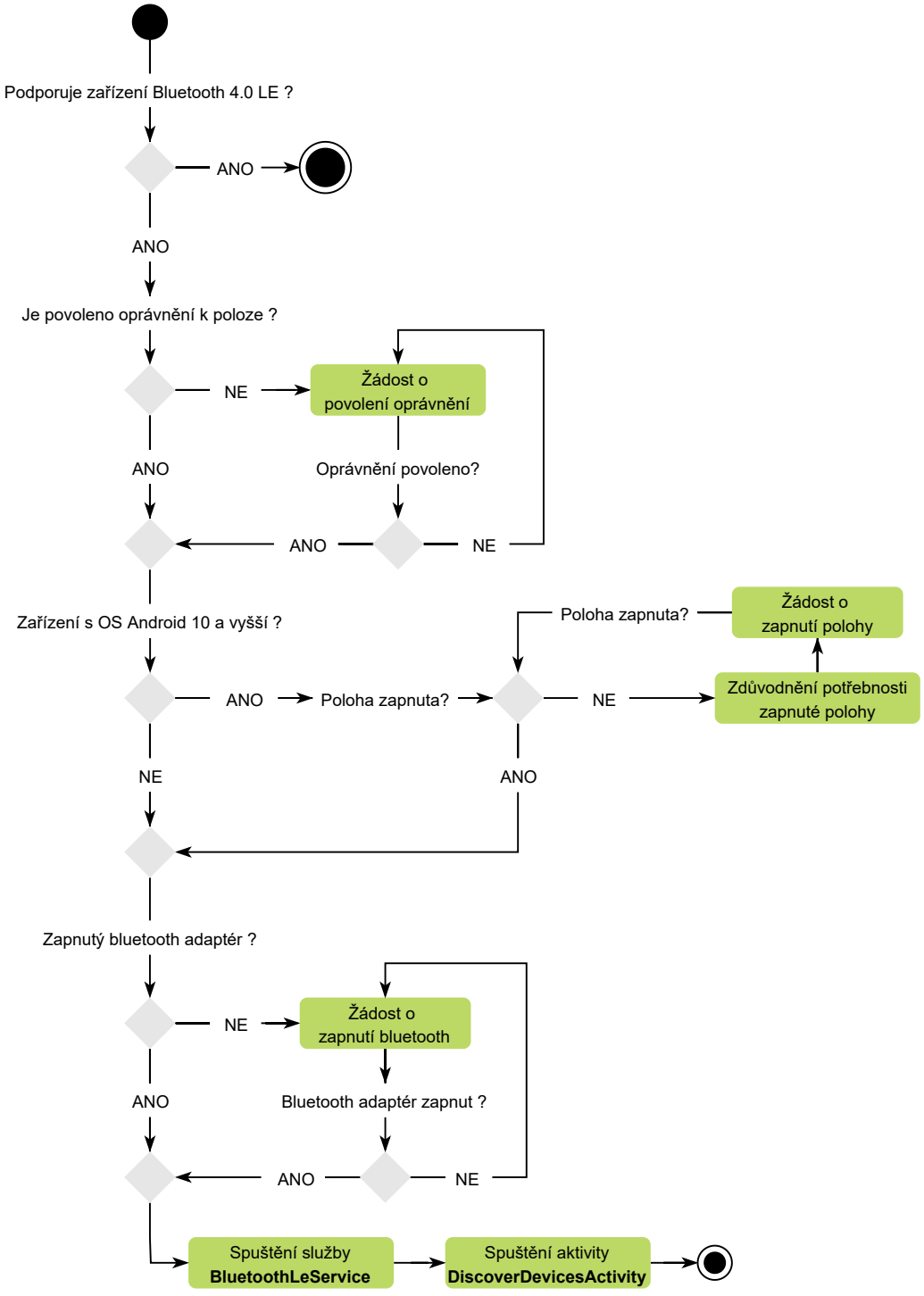
Druhou z podmínek pro pokračování aplikace je kontrola oprávnění. Aplikace vyžaduje pouze jedno oprávnění – přístup k poloze, které musí být uděleno a uživatel jej musí odsouhlasit. Zařízení s technologií Bluetooth 4.0. LE mohou na základě RSSI a vysílacího výkonu zjistit docela přesnou polohu mobilního zařízení, takže je logické, že by měl uživatel souhlasit. [26] Při úplně prvním

spuštění aplikace je ihned požádáno o udělení oprávnění k přístupu k poloze. Pokud uživatel oprávnění povolí je celá procedura získání oprávnění ukončena a aktivita může dále pokračovat. Jestliže uživatel oprávnění neudělí, je mu zdůvodněn důvod tohoto požadavku, který byl již popsán výše, a je znova o povolení oprávnění k přístupu k poloze požádán. Pokud uživatel oprávnění definitivně zamítne, je mu vysvětleno jeho rozhodnutí a nabídnuto přesměrování do systémového nastavení, kde může svůj nesouhlas odvolat. V opačném případě dojde k ukončení aplikace. Na obrázku 5.3 je celý podproces získání povolení k přístupu k poloze znázorněn detailněji.

Po kontrole oprávnění provádí aktivita kontrolu zapnutého adaptéru polohy, ale pouze u zařízení s verzí operačního sytému Android 10 a vyšší. Jedná se o další, zpřísněnou ochranu uživatele, která se poprvé objevila v této verzi. [27] Pokud uživatel v době kontroly nemá zapnutou polohu, je mu vysvětleno, proč musí být zapnuta a následně je přesměrován do nastavení, kde může polohu zapnout. Po návratu z nastavení aktivita zkontroluje, jestli byla poloha ve skutečnosti zapnuta. Pokud ne, zobrazí se vysvětlení znovu. Pokud ano, je stejně jako v případě, že poloha byla zapnuta již před samotnou kontrolu přikročeno k dalšímu kroku. Pokud se jedná o zařízení s verzí operačního systému Android menší než 10, je krok kontroly zapnutého adaptéru polohy přeskočen.

Předposledním krokem v kontrole požadavků na aplikaci je kontrola zapnutého Bluetooth adaptéru. Tento požadavek není předem nikterak uživateli vysvětlen, jelikož se jeho potřeba vzhledem k povaze aplikace pokládá za samozřejmý.

Jsou-li výše popsané požadavky splněny, spustí aktivita službu BluetoothLeService a následně i aktivitu DiscoverDevicesActivity. Poté je aktivita StartUpActivity ukončena a uživatelský děj se přenese do nově spuštěné aktivity. Celý algoritmus aktivity StartUpActivity je znázorněn na obrázku 5.4

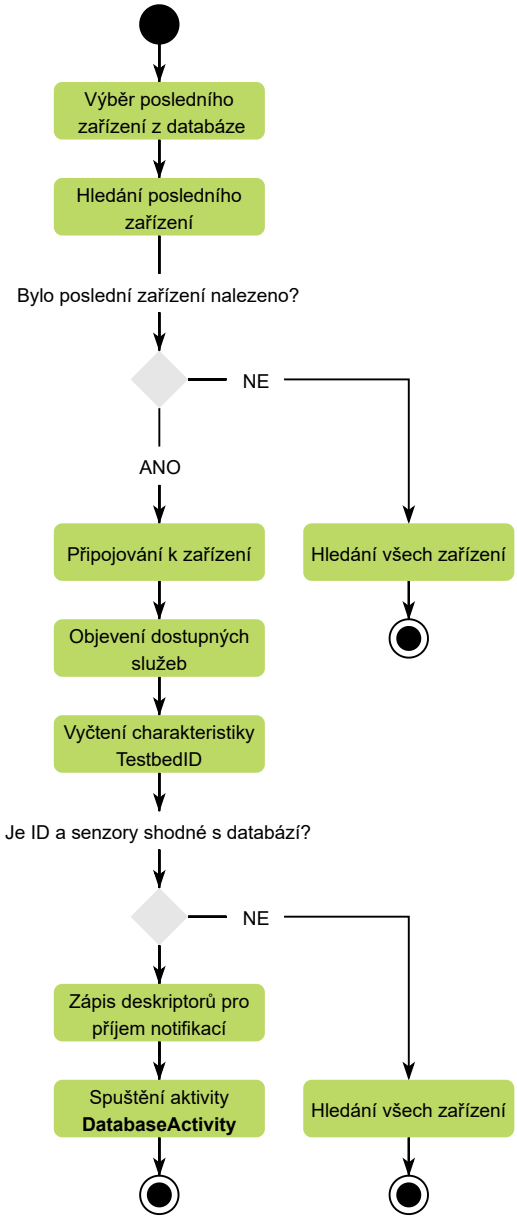


Obrázek 5.4: Diagram aktivit povolení oprávnění k poloze, zdroj: autor

5.2.2 DiscoverDevicesActivity

Druhou aktivitou aplikace je aktivita DiscoverDatabaseActivity. Jejím úkolem je provést vyhledání dostupných zařízení a zjištění jejich ID a senzory, kterými je daný produkt testbed osazen.

Jedná-li se o první spuštění aktivity v rámci běhu aplikace, provádí aktivita jinou sekvenci procesů než v případě, že se jedná o spuštění opakované. V případě prvního spuštění je cíleně hledáno zařízení, ke kterému byla aplikace naposledy připojena. V případě, že se jedná o spuštění opakované, jsou hledány všechny produkty testbed v dosahu mobilního zařízení.



Obrázek 5.5: Diagram aktivit vyhledání posledního připojeného zařízení, zdroj: autor

Při prvním spuštění je z lokální databáze vybrán testbed, ke kterému byla aplikace naposledy připojena. Parametry (název zařízení, MAC adresa) jsou nastaveny jako filtry při zahájeni vyhledávání tohoto testbedu v dosahu mobilního zařízení. Pokud není zařízení do určité doby nalezeno, je celý proces přerušen a aktivita pokračuje ve standardním necíleném vyhledávání. V případě, že se v daném časovém intervalu podaří zařízení nalézt, je zahájeno připojování k danému zařízení. Po navázaní spojení jsou pak zjištěny dostupné služby daného produktu testbed. Ze služby (Testbed Profile) určené pro sdílení dat s okolím je vyčtena hodnota charakteristiky (TestbedID). Tato hodnota je posléze porovnána s dříve získaným záznamem v databázi. V případě shody ID a osazených senzorů jsou zapsány hodnoty patřičných deskriptorů pro nastavení přijímání notifikací z produktu testbed. Aktivita je tímto ukončena, a je spuštěna další aktivita, DatabaseActivity. V případě, že se ID nebo osazené senzory neshodují, je stejně jako v případě neúspěšného vyhledaní zařízení přistoupeno k necílenému vyhledaní ostatních zařízení.

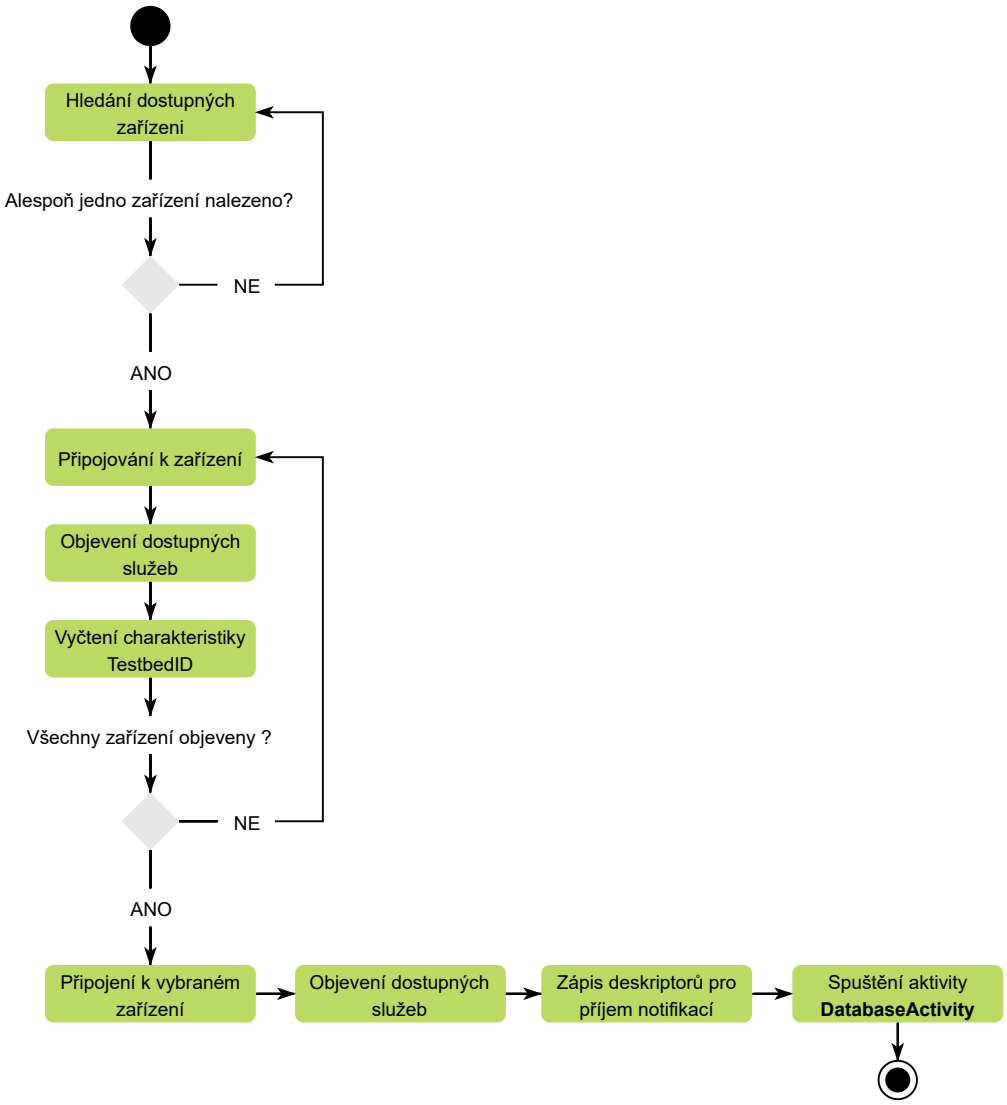
První spuštění této aktivity a tím pádem i spuštění výše uvedeného procesu může provést pouze aktivita StartUpActivity. Na obrázku 5.5 je diagram znázorňující proces během prvního spuštění aktivity.

Jakékoli jiné spuštění nebo návrat do aktivity vede automaticky ke spuštění procesu pro vyhledání všech dostupných produktů testbed bez ohledu na jejich záznam v lokální databázi mobilního zařízení. Tento proces je typicky spuštěn při návratu do aktivity z jiné aplikace, nebo při odpojení od zařízení.

Nejdříve je spuštěno hledání všech dostupných produktů testbed, tedy všech Bluetooth Low Energy zařízení s patřičným názvem. Vyhledávání trvá stejně jako v předchozím případě určitou dobu, jelikož samotné hledaní je velmi energeticky náročné. Po uplynutí daného intervalu se aktivita postupně připojuje ke všem nalezeným zařízením. Po navázáni spojení jsou opět zjištěny dostupné služby daného produktu testbed a následně vyčtena charakteristika (TestbedID) ze služby (Testbed Profile). Po připojení o prozkoumání všech nalezených zařízení je vyčkáváno, dokud uživatel nezvolí, ke kterému zařízení se má aplikace připojit. Po vybrání zařízení proběhne navázání spojení s vybraným produktem testbed, objevení dostupných služeb a zapsaní hodnot do patřičných deskriptorů pro povolení přijímání notifikací. Následně dojde k ukončení aktivity a spuštění aktivity DatabaseActivity. Na obrázku 5.6 je znázorněn pomocí diagramu aktivit proces vyhledání a objevení všech dostupných zařízení.

5.2.3 DatabaseActivity

Hlavní aktivitou celé aplikace je aktivita DatabaseActivity. Tato aktivita je spuštěna po připojení aplikace k produktu testbed a jeho přípravě na zasílání dat. Aktivita samotná nevykonává žádný předem definovaný proces, ale reaguje na jednotlivé události a vnější podněty uživatele. Aplikace se skládá ze tří fragmentů, které nabízí uživateli různé pohledy na prezentovaná data. Uživatel může zároveň v této aktivitě přepínat mezi typem prezentovaných dat a měnit časový interval, ve kterém



Obrázek 5.6: Diagram aktivit vyhledání posledního připojeného zařízení, zdroj: autor

chce data zobrazit. Dále má uživatel možnost aplikaci ukončit, odpojit se od připojeného produktu testbed a zobrazit informace o aplikaci.

5.2.4 BluetoothLeService

Na pozadí celé aplikace pracuje služba s názvem BluetoothLeSerivce. Jak již ze samotného názvu vyplývá, úkolem služby je komunikace s produktem testbed prostřednictvím technologie Bluetooth 4.0 Low Energy. Použití služby místo aktivity se v tomto případě přímo nabízí.

Pro připojení k produktu testbed pomocí Bluetooth Low Energy je třeba vytvořit instanci GATT klienta. Ten naváže spojení s GATT serverem, kterým je právě testbed. GATT klient je využíván jednak v aktivitě DiscoverDevicesActivity pro připojení a objevení jednotlivých zařízení a také v ak-

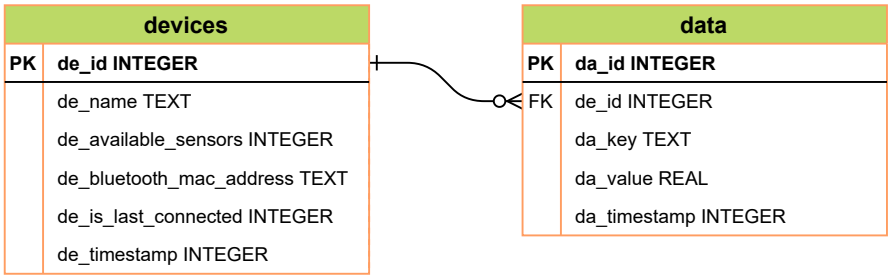
tivitě DatabaseActivity pro přijímání dat. Použití služby tak jednak logicky odděluje implementaci GATT klienta od zbytku aplikace, a jednak se tím dá elegantně vyhnout použití dalšího vlákna pro asynchronní proces, jak by tomu muselo být v případě implementace klienta přímo v aktivitě.

Navíc je předpoklad, že data budou vysílána asynchronně a v delším časovém intervalu. Implementace GATT klienta v aktivitě by tak vyžadovala neustále aktivní aktivitu, která by nesměla být překryta ani pozastavena, aby mohla data přijímat. Služba, která běží na pozadí může přijímat data tak dlouho, dokud není služba zastavena.

Úkolem služby je na úkon některých z připojených aktivit navazování spojení s GATT serverem, objevování dostupných služeb a zapisování hodnot do deskriptorů jednotlivých charakteristik. Dále služba znovu navazuje spojení s posledním připojeným zařízením v případě náhlého výpadku spojení s GATT serverem. Služba také do připojené aktivity zasílá zprávy o nově dostupných datech, která rovnou ukládá do lokální databáze.

5.3 Návrh databáze

Pro ukládání jednotlivých záznamů a zařízení má aplikace k dispozici lokální databázi. K ukládání nenáročných a jednoduchých strukturovaných dat je vhodný databázový systém SQLite, který oproti pokročilým databázovým systémům využívá pouze jednoduché datové typy. Systém SQLite navíc nabízí nativní podporu na zařízeních s operačním systémem Andorid skrze původní API.



Obrázek 5.7: ER diagram struktury lokální databáze, zdroj: autor

Databáze se skládá ze dvou tabulek. První pro ukládání známých produktů testbed (`devices`) a druhá k ukládání jednotlivých záznamů (`data`). Mezi tabulkou `devices` a `data` je vazba 1:N. Platí tedy, že každý záznam v tabulce `devices` má referenci na více záznamů v tabulce `data`.

Tabulka `devices` má celkem šest sloupců. Sloupec `de_id` s celočíselným datovým typem je primárním klíčem celé tabulky. Hodnota primárního klíče je shodná s ID produktu testbed. Sloupce `de_name` a `de_bluetooth_mac_address` s textovými datovými typy nesou údaje o názvu zařízení a jeho MAC adrese. Ve sloupci `de_available_sensors` s celočíselným datový typem je uložena informace o osazených senzorech daného testbedu. Předposlední sloupec `de_is_last_connected` s celočíselným datovým typem nese údaj o posledním připojeném zařízení, sloupec `de_timestamp` pak obsahuje časový údaj o pořízení konkrétního záznamu.

Druhá tabulka `data` má pět sloupců. Sloupec `da_id` s celočíselným datovým typem je primárním klíčem této tabulky. Jeho hodnota se automaticky inkrementuje s každým dalším záznamem. Druhý sloupec `de_id` s celočíselným datový typem je cizím klíčem s referencí na záznam v tabulce `devices`. Sloupec `da_key` obsahuje textovou hodnotu, která definuje typ záznamu. Samotná hodnota je v reálném datovém typu uložena ve sloupci `da_value`. Poslední sloupec `da_timestamp` pak obsahuje údaj v podobě časového razítka o pořízení daného záznamu.

Na obrázku 5.7 je prostřednictvím ER diagramu znázorněna struktura lokální databáze včetně vazeb mezi tabulkami a datového slovníku.

Kapitola 6

Implementace

Z hlediska plánovaného využití aplikace bylo rozhodnuto pro nativní vývoj aplikace pro operační systém Android. Implementace probíhala ve volně dostupném vývojovém prostředí (IDE) na platformě IntelliJ® – Android Studio 4.1.2.

Android Studio je ucelené prostředí, které zahrnuje editor, kompilátor a ladicí prostředek pro zařízení s operačním systémem Android. Kromě spuštění a ladění vyvíjené aplikace na reálném zařízení umožňuje prostředí také simulaci na integrovaném emulátoru. Prostředí rovněž nabízí nástroj pro měření výkonu, využití baterie nebo sítě mobilního zařízení v reálném čase.

Během vývoje byl celý projekt aplikace zálohován a verzován pomocí verzovacího nástroje GIT.

6.1 Implementace UI

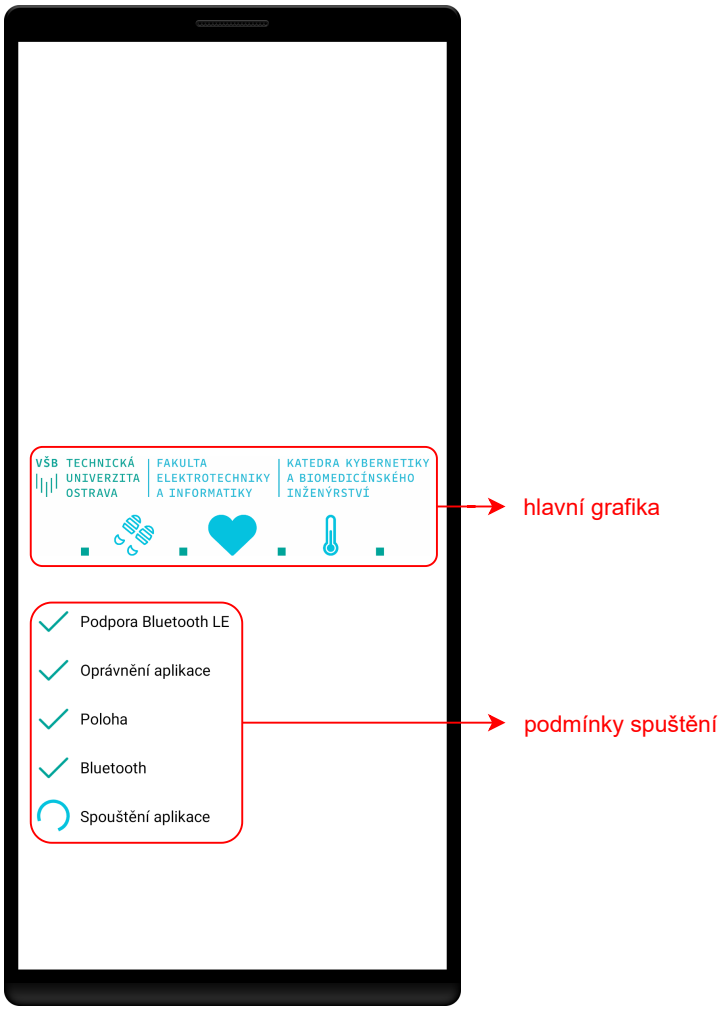
První implementovanou částí bylo uživatelského rozhraní. Android Studio umožňuje přímý grafický návrh front-endové části za použití základních prvků (textová pole, seznamy, tlačítka, výběrové boxy, atd.) bez nutnosti skládat návrh jako jednotlivé atributy v jazyce XML. Grafický editor průběžné během návrhu překládá a vytváří zdrojový XML soubor.

Grafický styl celé aplikace je stylizován do barev, které odpovídají vizuálnímu stylu univerzity a fakulty. [28] Hlavními barvami jsou zelená s hodnotou `#00A499` a tyrkysová s hodnotou `#05C3DE`.

6.1.1 UI aktivity StartUpActivity

Uživatelské rozhraní aktivity StartUpActivity je založeno na pružném typu layoutu (Constraint Layout). To znamená, že každý prvek je vztažen k jinému prvku nebo okraji obrazovky pomocí fixních nebo plovoucích referencí.

ImageView pro obrázek hlavní grafiky je zafixován ke všem čtyřem stranám obrazovky s minimální vzdálenosti od každého okraje 16 dp. Tento typ layoutu tak zajišťuje, že na jakkoliv velkém zařízení bude hlavní grafika stále vycentrovaná ve středu obrazovky s minimálním odsazením od okraje.



Obrázek 6.1: UI aktivity StartUpActivity, zdroj: autor

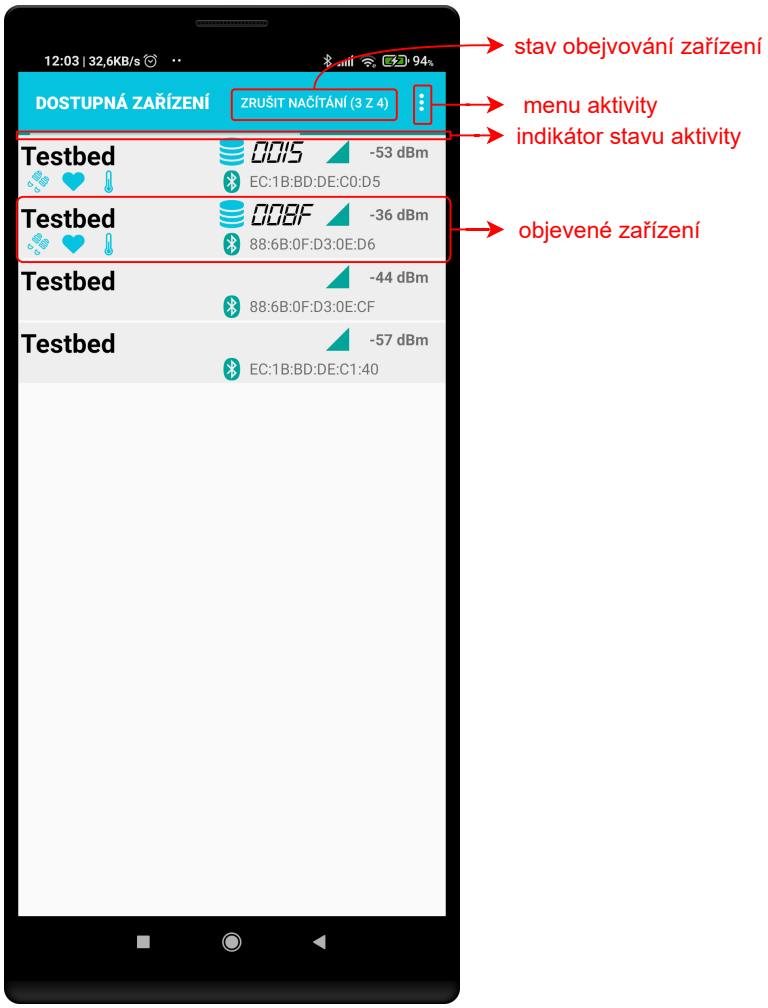
Pod hlavní grafikou je umístěn seznam položek (ListView). Každá položka seznamu se vztahuje ke kontrole jedné z podmínek, které jsou nutné pro spuštění aplikace.

Aktivita je navržena přes celou obrazovku, nezobrazuje se ani stavový řádek v horní části obrazovky, ani navigační tlačítka zařízení ve spodní části obrazovky. Orientace UI je pevně zafixována na horizontální polohu. Podoba UI této aktivity je na obrázku 6.1.

6.1.2 UI aktivity DiscoverDevicesActivity

Druhá aktivita v pořadí, DiscoverDevicesActivity, je opět založena na pružném layoutu. Aktivita již neokupuje celou obrazovku, ale ponechává patřičné místo hornímu stavovému řádku a také spodním navigaci.

V horní části aktivity je umístěn tyrkysový panel, který obsahuje titulek, stav objevování zařízení a tlačítko pro rozbalení menu aktivity. Kliknutím na text se stavem objevování může uživatel objevo-

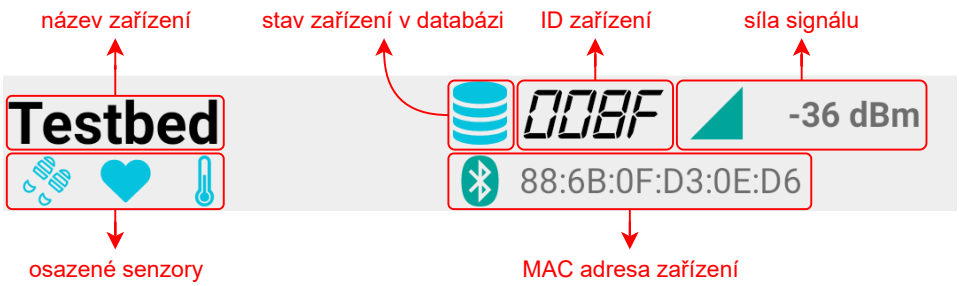


Obrázek 6.2: UI aktivity DiscoverDevicesActivity, zdroj: autor

vání zastavit, nebo znova spustit. Tlačítkem s třemi tečkami pak uživatel vyvolá rozbalovací menu, které mu umožní zobrazit informace o aplikaci, nebo ji ukončit. Zbytek rozložení tvoří seznam položek s vlastním layoutem. Kliknutím na položku ze seznamu uživatel zahájí proceduru připojování k objevenému zařízení. Na obrázku 6.2 je vyobrazena podoba UI aktivity DiscoverDevicesActivity.

Layout, kterým je tvořen seznam zařízení je lineární layout. Plocha, kterou jednotlivé prvky okupují je přímo úměrná hodnotě váhy jednotlivého prvku. Layout je tvořen jedním vertikální lineárním layoutem, který obsahuje dva horizontální lineární layouty (každý pro jeden řádek). V řádku jsou pak již naskládány jednotlivé prvky.

Název zařízení, síla signálu a MAC adresa zařízení jsou údaje známé z vyhledání produktů testbed jako klasických Bluetooth Low Energy zařízení. Jakmile je aplikace k zařízení připojena a dostupné služby jsou objeveny, mohou být ze zařízení vyčteny dodatečné informace. Dostupné senzory jsou prostřednictvím ikon zobrazeny pod názvem zařízení. Osazené senzory jsou indikovány



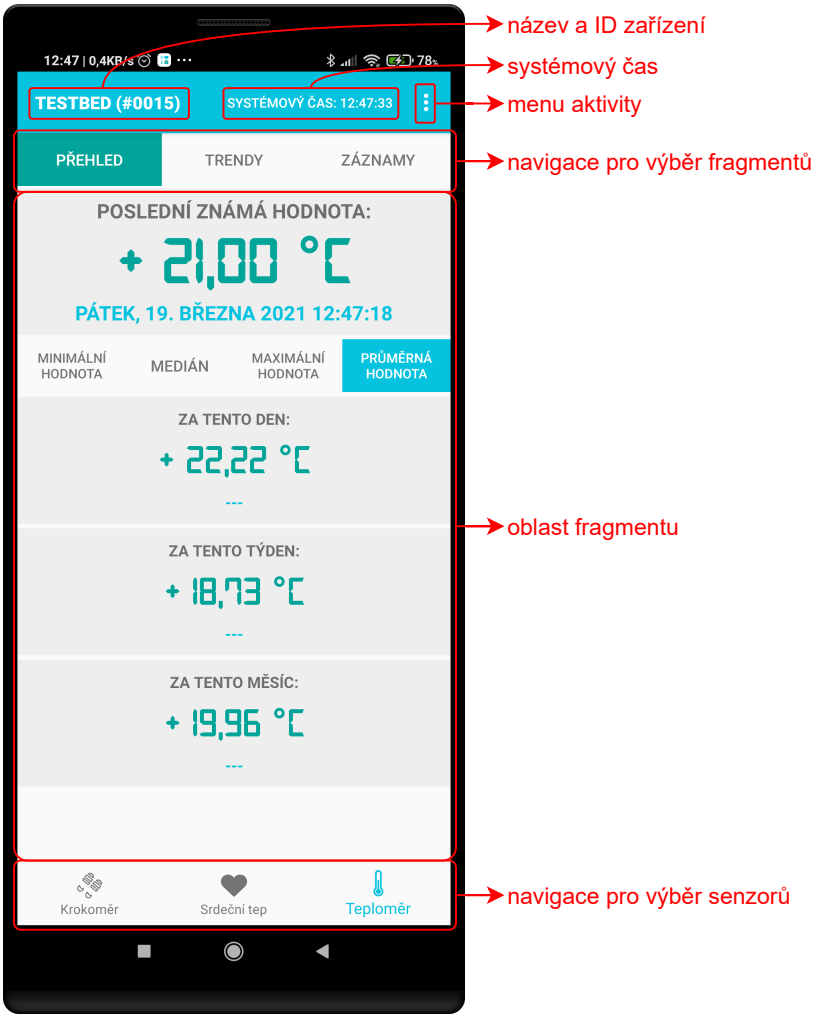
Obrázek 6.3: UI layoutu produktu testbed, zdroj: autor

tyrkysovou barvou, neosazené pak šedou. Nalevo od síly signálu je pak zobrazeno vyčtené ID. Před hodnotou ID je ikona indikující stav zařízení v databázi. Pokud je ikona tyrkysové barvy je zařízení zcela shodné se záznamem v databázi. Liší-li se vyčtené informace se záznamem v databázi v MAC adrese, nebo osazených senzorech zařízení, je ikona červené barvy. Neexistuje-li záznam v databázi k danému nalezenému zařízení, je ikona šedé barvy. Na obrázku 6.3 je layout ze seznamu zařízení vyobrazen.

6.1.3 UI aktivity DatabaseActivity

Aktivita DatabaseActivity je založena opět na pružném typu layoutu.

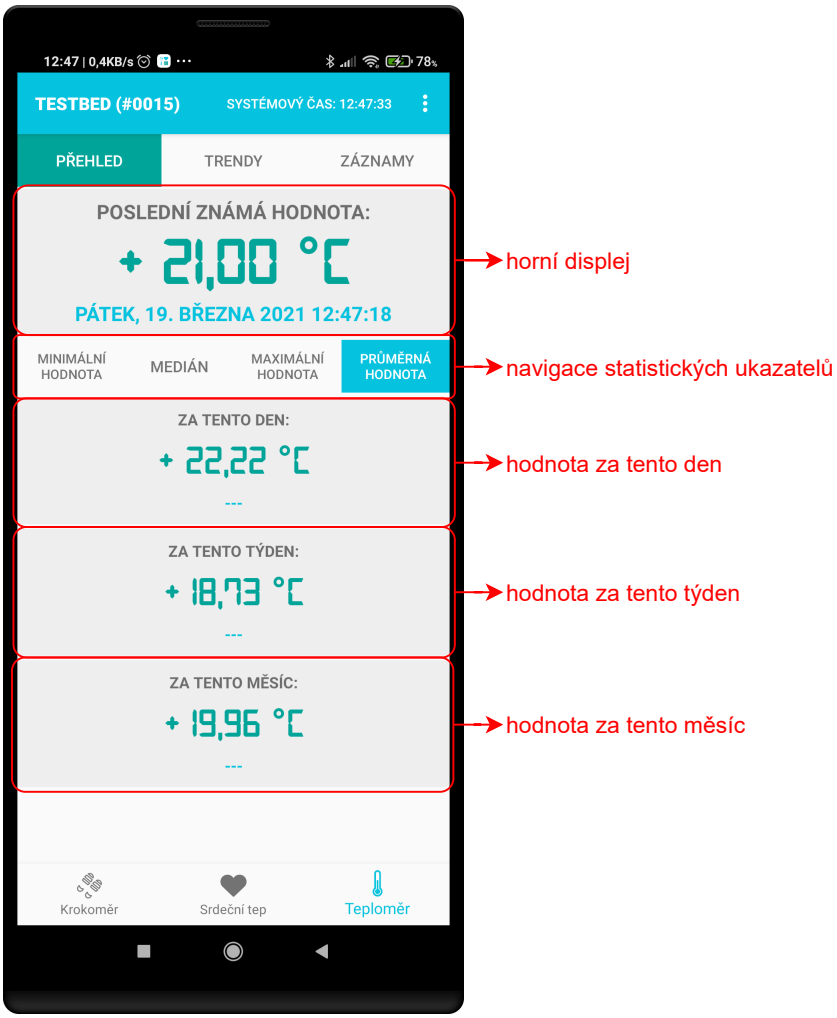
Horní část layoutu je stejně jako v případě aktivity DiscoverDevicesActivity ponechána tyrkysovému panelu, který obsahuje název a ID zařízení. Dále je zde s přesností na sekundy pravidelně aktualizován systémový čas pro přesnější srovnání záznamů uživatelem. Tlačítkem s třemi tečkami je vyvoláno rozbalovací menu, které oproti aktivitě DiscoverDevicesActivity nabízí navíc možnost odpojit se od právě připojeného zařízení a export dat do CSV. Pod tyrkysovým panelem se nachází navigace realizovaná pomocí prvku TabLayout. Součástí aktivity DatabaseActivity jsou rovněž tři fragmenty, které jsou přepínány vrchní navigací. Obsah těchto fragmentů je pak přepínán spodní navigací. Na obrázku 6.4 je vyobrazeno rozložení této aktivity.



Obrázek 6.4: UI aktivity DatabaseActivity, zdroj: autor

UI fragmentu OverviewFragment

První fragment slouží pro prezentaci rychlého přehledu uložených dat. Tento fragment je uživateli zobrazen jako první po připojení k produktu testbed.

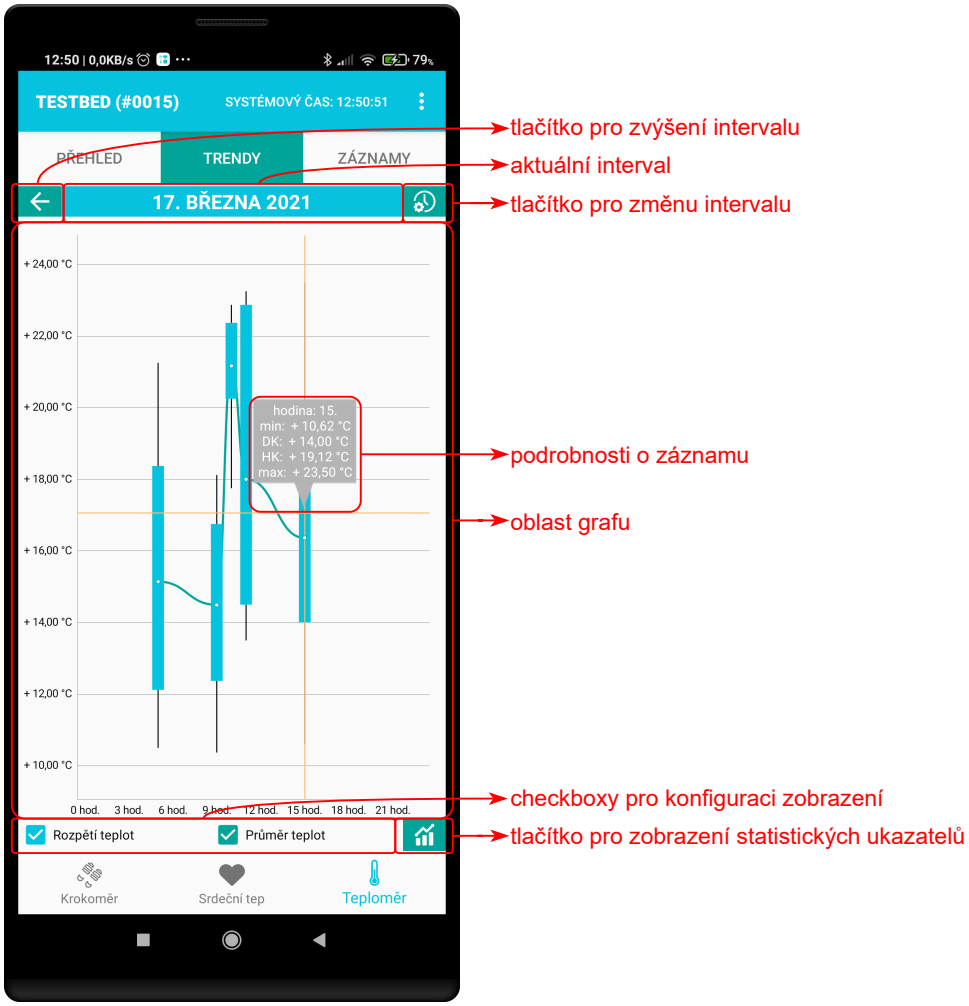


Obrázek 6.5: UI fragmentu OverviewFragment, zdroj: autor

Horní displej zobrazuje poslední známou hodnotu včetně časového razítka. Pod horním displejem se nachází další navigace, která nabízí uživateli na výběr zobrazovaného statistického ukazatele. V případě, že je vybrán krokoměr jako zdroj dat, má uživatel na výběr pouze z jednoho statistického ukazatele – celkový úhrn. U ostatních dat je na výběr mezi minimální, průměrnou a maximální hodnotou a také mediánem. Zbylé tři displeje poté zobrazují statistickou hodnotu za tento den, tento týden a také tento měsíc. U kvantilových hodnot je rovněž zobrazeno i datum a čas pořízení záznamu. Každá změna hodnoty je indikována krátkou změnou barvy textu. UI fragmentu OverviewFragment je vyobrazeno na obrázku 6.5.

UI fragmentu ChartFragment

Druhým ze třech fragmentů, které lze zobrazit prezentuje data v grafické podobě. Tato data lze zobrazit v základních časových intervalech jako je rok, měsíc, den v měsíci, hodina a minuta.

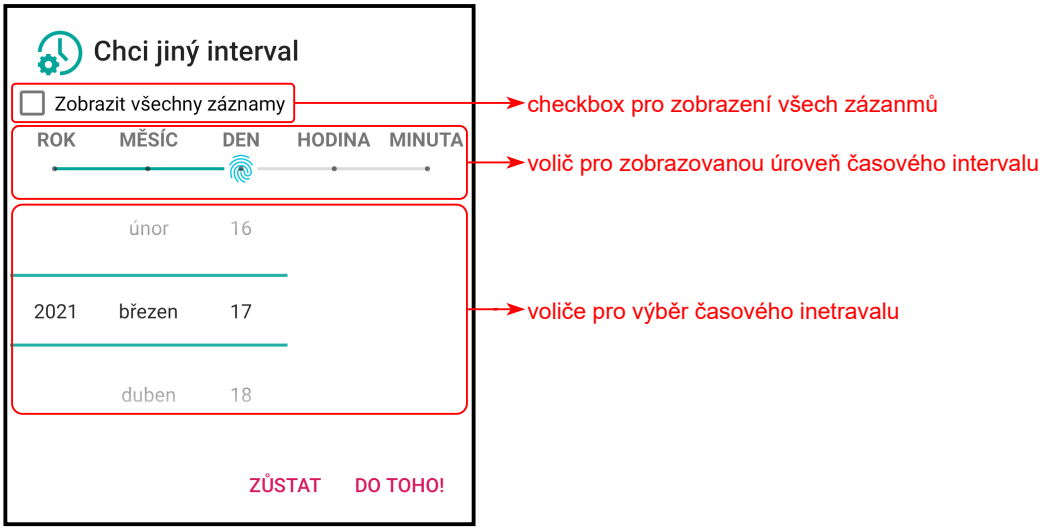


Obrázek 6.6: UI fragmentu ChartFragment, zdroj: autor

Aktuální interval, ze kterého jsou data prezentována je zobrazen v oblasti pod horní navigací. Nalevo od něj se nachází tlačítko, kterým lze zobrazovaný interval zvýšit. Tlačítkem napravo od aktuálně zobrazovaného intervalu lze zobrazit dialogové okno, které umožňuje přesně zvolit nový zobrazovaný interval. Prostřední část fragmentu okupuje graf. Svislá osa grafu nese hodnoty jednotlivých záznamů (kroky, tepy za minutu nebo °C), vodorovná osa pak rozděluje záznamy prezentované v grafu do jednotlivých časových intervalů. V případě srdečních tepů a teploty se jedná o krabicový graf. Jednotlivé záznamy v grafu pak zobrazují minimální a maximální hodnotu a také první a třetí kvartil. Jednotlivé záznamy jsou také proložené průměrnou hodnotou. V případě kroků je zobrazovaný graf sloupcový. Jednotlivé sloupce zobrazují pouze celkový úhrn kroků. Dotykem na záznam

v grafu se zobrazí informace o daném záznamu. Zobrazení záznamů v grafu lze konfigurovat pomocí checkboxů umístěnými pod grafy. Těmi lze jednotlivé záznamy grafu vypínat a zapínat. Tlačítkem v pravém spodním rohu lze zobrazit důležité statistické ukazatele za vybraný časový interval. Na obrázku 6.6 je vyobrazen fragment ChartFragment.

Dialogové okno pro nastavení časového intervalu umožňuje uživateli vybrat libovolnou úroveň a časový interval.



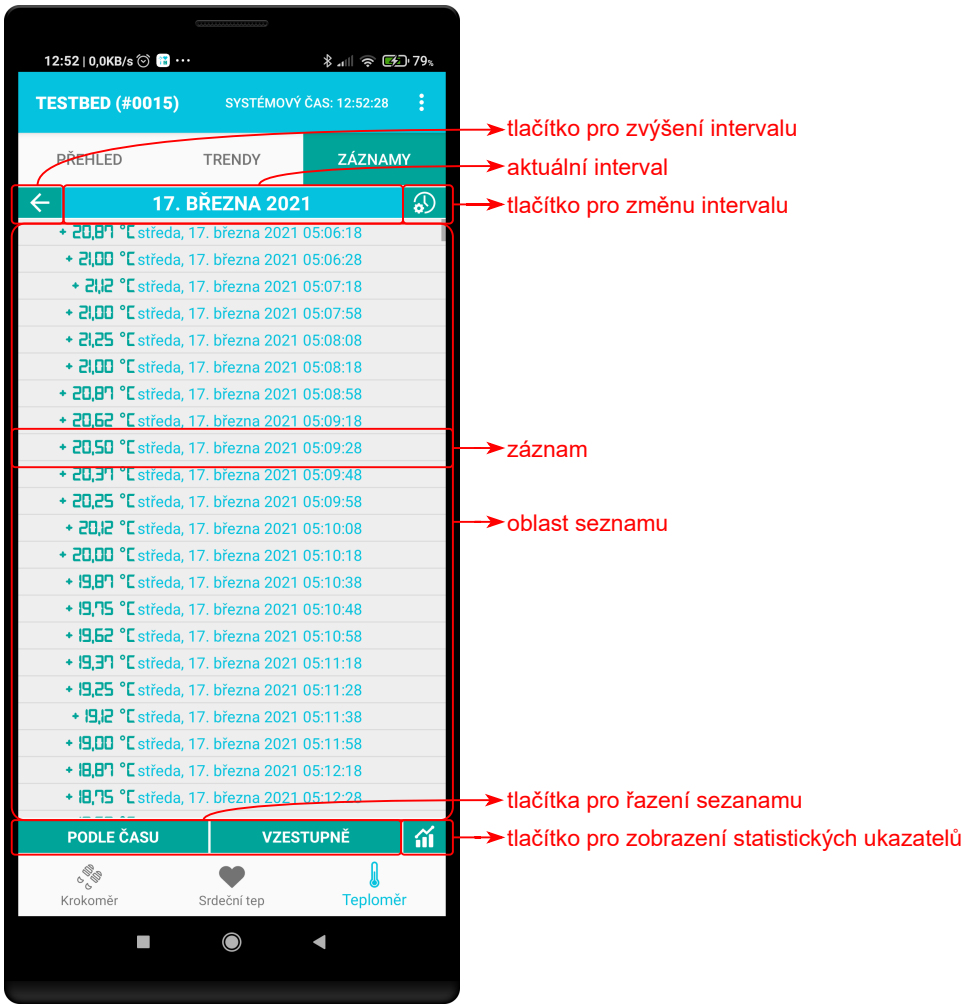
Obrázek 6.7: UI dialogového okna s voliči pro změnu časového intervalu, zdroj: autor

Horizontálním voličem s diskrétními hodnotami uživatel vybírá úroveň časového intervalu, čímž zároveň zpřístupňuje kruhové voliče pro nastavení roku, měsíce, dne v měsíci, hodiny a minuty. Checkbox v horní části dialogového okna umožňuje zobrazit všechny záznamy na úrovni jednotlivých roků. Vzhled dialogového okna je vyobrazen na obrázku 6.7

UI fragmentu ListFragment

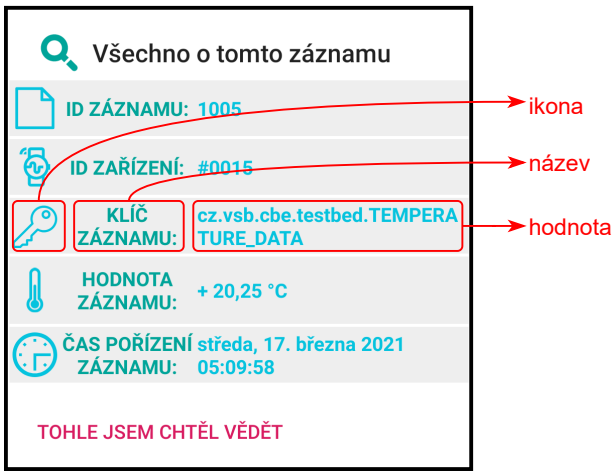
Poslední fragment prezentuje data v podobě jednotlivých záznamů. Stejně jako v případě fragmentu ChartFragment lze data zobrazit v časových intervalech.

Aktuální interval je zobrazen pod horní navigací. Tlačítka po stranách intervalu slouží stejně jako u fragmentu ChartFragment ke zvýšení a změně aktuálního intervalu. Největší část fragmentu pod intervalem okupuje seznam s jednotlivými záznamy. Ten zobrazuje hodnotu a časové razítko záznamu. Dotykem na záznam lze zobrazit dialogové okno s podrobnostmi záznamu. Dlouhý dotyk na záznam poté sníží aktuální časový interval o jednu úroveň. Ve spodní části fragmentu je umístěna dvojice tlačítek pro řazení záznamu. Záznamy lze řadit vzestupně nebo sestupně, a to podle hodnoty nebo časového razítka. V pravém spodním rohu je tlačítko pro zobrazení důležitých statistických ukazatelů. Vzhled fragmentu pro zobrazení jednotlivých záznamů je na obrázku 6.8



Obrázek 6.8: UI fragmentu ListFragment, zdroj: autor

Dialogové okno s detaily záznamu je složeno ze seznamu, kde jednotlivými položkami jsou vlastnosti vybraného záznamu. Každá hodnota je uvozena názvem a pro lepší ilustraci a rychlejší orientaci také ikonou. Příklad dialogového okna s detaily záznamu je vyobrazen na obrázku 6.9.



Obrázek 6.9: UI dialogového okna s detaily záznamu, zdroj: autor

6.2 Implementace back-endové části

Po front-endové části bylo přikročeno k implementaci obslužného kódu aplikace. Obslužný kód je napsán v objektově orientovaném jazyce Java s použitím API pro operační systém Android. Aplikace API verze 23, což znamená, že je aplikace kompatibilní s verzí operačního systému Android 6.0 a vyšší. Zdrojový kód aplikace je v rámci balíčku rozdělen do šesti pod-balíčků:

- balíček `activitiesAndServices` obsahuje zdrojové kódy všech aktivit a služeb použitých v aplikaci
- balíček `adapters` obsahuje zdrojové kódy pro seznamy použité v aplikaci
- Balíček `chart` obsahuje zdrojové kódy pro vykreslení grafů a ukazatelů na jednotlivé záznamy
- balíček `fragments` obsahuje zdrojové kódy tří fragmentů, které se vykreslují v prostoru aktivity DatabaseActivity
- balíček `sql` obsahuje zdrojové kódy pro práci s databází, zařízeními a záznamy
- balíček `utils` obsahuje třídy a metody pro formátování hodnot, výpočet statistických ukazatelů nebo UUID identifikátory služeb a charakteristik Bluetooth Low Energy zařízení

6.2.1 Implementace aktivity StartUpActivity

Aktivita StartUpActivity je potomkem třídy AppCompatActivity, od které dědí veškeré systémové metody. Celý proces spuštění aplikace je rozdělen do několika samostatných metod, které jsou volány systematicky po ověření splnění některé z podmínek pro spuštění aplikace.

Po startu aktivity je nejdříve operačním systémem volána metoda `onCreate()`. V této metodě, je nejprve nastavena orientace obrazovky a celoobrazovkový režim zobrazení. Dále je získán odkaz na seznam umístěný pod hlavní grafikou. Dalším krokem je ověření podpory Bluetooth Low Energy. Pokud není tato technologie mobilním zařízením podporována, je uživateli zobrazeno prostřednictvím dialogového okna upozornění a aplikace následně ukončena. V opačném případě je získána závislost Bluetooth adaptéru a voláním metody `permissionAndExplanationProcedure()` pokračuje proces spuštění aplikace.

Tato metoda kontroluje stav potřebných oprávnění a potřebu jejich řádného zdůvodnění. V případě, že oprávnění nebylo dosud uděleno, zobrazí aktivita v dialogovém okně žádost o udělení potřebného oprávnění. Po uživatelské interakci s dialogovým oknem je aktivitou volán callback `onRequestPermissionsResult()`. Obslužný kód tohoto callbacku zjišťuje výsledek udělení oprávnění. Pokud bylo udělení oprávnění zamítnuto, nebo bylo odebráno, je zobrazeno dialogové okno s vysvětlením potřebnosti daného oprávnění pro chod aplikace. Po zjištění povolených oprávnění je zavolána metoda `enableLocation()`.

V případě, že je verze operačního systému vyšší než Android 10, ověřuje metoda zapnutou polohu zařízení. Pokud je poloha vypnutá, zobrazí aktivita uživateli dialogové okno, kde jej informuje o potřebnosti zapnutého adaptéru. Posléze je aktivita přerušena a je otevřeno systémové nastavení, kde může uživatel polohu zařízení zapnout. Po návratu zpátky do aplikace je znova volána metoda `enableLocation()`, která buď znova zobrazí dialogové okno, pokud uživatel polohu nezapnul, nebo zavolá další metodu `enableBluetoothAdapter()`, která pokračuje v procesu spuštění aplikace.

Metoda `enableBluetoothAdapter()` kontroluje zapnutí Bluetooth adaptéru. V případě vypnutého adaptéru je uživateli zobrazeno systémové dialogové okno s žádostí aplikace o zapnutí adaptéru. V opačném případě je volána metoda `continueApplication()`.

```
mOnBluetoothAdapterEnabledActivityResult.launch(
        new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE));
mStartUpConditionsAdapter.setCondition(
        new StartUpConditionsOrder().BLUETOOTH_ADAPTER_STATUS,
        ConditionsAdapter.IN_PROGRESS,
        getString(R.string.activity_start_up_condition_bluetooth_adapter));
```

Výpis zdrojového kódu 6.1: Žádost o zapnutí Bluetooth adaptéru

Globální proměnná `mOnBluetoothAdapterEnabledActivityResult` je instancí systémové třídy `ActivityResultLauncher<Intent>`. Instancializace této třídy probíhá pomocí systémové metody `registerForActivityResult()`, která definuje callback, jenž je vykonán po skončení aktivity spuštěné metodu `launch()`. Na výpisu zdrojového kódu 6.1 je ukázka vyvolání systémového dialogového okna s žádostí o zapnutí Bluetooth adaptéru.

Poslední metoda v procesu `continueApplication()` spustí službu BluetoothLeService. Poté pošle příznak do nadcházející aktivity DiscoverDevicesActivity, že se jedná o první běh aplikace a tuto aktivitu spustí. Aktivita StartUpActivity je poté ukončena.

6.2.2 Implementace aktivity DiscoverDevicesActivity

Tato aktivita je opět potomkem třídy AppCompatActivity, od které dědí veškeré systémové metody. V metodě `onCreate()`, volané po startu aktivity, je nastavena orientace obrazovky. Dále jsou naplněny instance pro práci s databází, Bluetooth adaptérem a Bluetooth skenerem. Posléze jsou získány odkazy na seznam, který bude obsahovat nalezená zařízení, a další prvky UI indikující aktuální stav aplikace. Tato metoda také vytváří dialogová okna, která jsou uživateli zobrazena během připojování aplikace k zařízení, nebo v případě vypnutí adaptérů polohy nebo bluetooh.

Po skončení metody `onCreate()` je operačním systémem přirozeně volána metoda `onResume()`. V těle této metody se aktivita připojuje ke službě BluetoothLeService. Po úspěšném navázaní spojení je volán callback `onServiceConnected()` proměnné `mBluetoothLeServiceConnection`. Callback získává instanci služby BluetoothLeService bežící na pozadí a zároveň inicializuje Bluetooth adaptér. Jako poslední krok metoda `onResume()` zahajuje skenování dostupných zařízení, a to na základě

hodnoty globální proměně `mFirstActivityRun`. V případě, že je hodnota této proměnné pravdivá, jedná se o první běh aplikace a je spuštěno cílené vyhledávání po posledním připojeném zařízení. V opačném případě je spuštěno vyhledávání po všech ostatních produktech testbed.

Spuštění skenování všech produktů testbed se provede voláním metody `scanTestbedDevices()`. Metoda na základě argumentu může zahájit, zastavit skenování nebo zrušit skenování. V případě zahájení je zkontrolován adaptér polohy (pouze u zařízení s verzí operačního systému 10 a vyšší), následně je zviditelněn text informující o nalezených zařízeních a indikátor probíhajícího vyhledávání. Seznam dříve nalezených zařízení je vymazán a do instance Bluetooth skeneru jsou vloženy parametry a callback pro vyhledávání zařízení. Po spuštění vyhledáváni je rovněž odstartován časovač, který počítá uplynulou dobu vyhledávání a po dosažení nastavené hodnoty (10 sekund) vyhledávání zastaví. V případě nálezu zařízení je volána metoda `onScanResult()` instance `mScanTestbedDevicesCallback`, která byla předána při zahájení vyhledávání Bluetooth skeneru. V těle metody je přidáno/aktualizováno zařízení v seznamu nalezených produktů testbed. Po přetečení časovače nebo uživatelském zásahu je skenování zastaveno. Pokud nebylo během vyhledávání žádné zařízení nalezeno, nebo bylo vyhledávání zrušeno, je vymazán seznam zařízení a zviditelněn text informující o nenalezených zařízení. Aktivita je poté připravena na zahájení nového vyhledávání.

Po dokončení vyhledávání se aplikace postupně připojuje ke všem nalezeným zařízením za účelem zjištění jejich ID a osazených senzorů. Připojení k prvnímu zařízení společně s registrací přijímače asynchronních událostí ze služby běžící na pozadí aplikace (BluetoothLeService) bylo provedeno v metodě `scanTestbedDevices()` po zastavení skenování. Metoda `onReceive()` instance přijímače `mDicovoveringTestbedDevicesBroadcastReceiver` je volána systémem, a to na základě zaslané události ze služby, se kterou byl přijímač svázán. V těle metody je přepínač (switch), který svými případy odpovídá jednotlivým stavům, které nastávají při připojování k Bluetooth zařízení.

```
public void onReceive(Context context, Intent intent) {
    switch (intent.getAction()) {
        case BluetoothLeService.ACTION_GATT_CONNECTED:
            mBluetoothLeService.discoverServices();
            break;

        case BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED:
            for (BluetoothGattService bluetoothGattService :
                    mBluetoothLeService.getSupportedGattServices()) {
                for (BluetoothGattCharacteristic bluetoothGattCharacteristic :
                        bluetoothGattService.getCharacteristics()) {
                    if (bluetoothGattCharacteristic.getUuid().equals(
                            SampleGattAttributes.TESTBED_ID_CHARACTERISTIC_UUID)) {
                        mBluetoothLeService.readCharacteristic(bluetoothGattCharacteristic);
                    }
                }
            }
```

```
            break;
```

Výpis zdrojového kódu 6.2: Přepínač uvnitř přijímače asynchronních zpráv – 1. část

Případ `BluetoothLeService.ACTION_GATT_CONNECTED` je vykonán při příjmu události informující o připojení k Bluetooth zařízení. V těle případu je spuštěno objevení dostupných služeb. Případ `BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED` je spojen s události indikující dokončení objevovaní služeb. V těle tohoto případu jsou charakteristiky napříč dostupnými službami dle UUID procházeny za účelem nalezení charakteristiky s hodnotou ID produktu testbed a informací o osazených senzorech. Tato charakteristika je následně vyčtena. Výpis zdrojového kódu 6.2 s popsanými případy je uveden výše.

```
        case BluetoothLeService.ACTION_TESTBED_ID_DATA_AVAILABLE:
            final TestbedDevice discoveringTestbedDevice = mDiscoveredTestbedDevicesAdapter
                    .getTestbedDevice(mDiscoveringDeviceIndex);
            discoveringTestbedDevice.setDeviceId(intent.getIntExtra(
                    BluetoothLeService.TESTBED_ID_DATA, 0x1FFFF));
            discoveringTestbedDevice.setAvailableSensors(intent.getIntExtra(
                    BluetoothLeService.AVAILABLE_SENSORS_DATA, 8));
            discoveringTestbedDevice.deviceWasDiscovered();
            discoveringTestbedDevice.setStoredState(
                    mTestbedDatabase.getStoredStatusOfTestbedDevice(
                            discoveringTestbedDevice));
            mDiscoveredTestbedDevicesAdapter.setTestbedDevice(discoveringTestbedDevice);
            mDiscoveredTestbedDevicesAdapter.notifyDataSetChanged();
            mBluetoothLeService.disconnect();
            break;

        case BluetoothLeService.ACTION_GATT_DISCONNECTED:
            if (mDiscoveredTestbedDevicesAdapter.getTestbedDevice(mDiscoveringDeviceIndex)
                    .isTestbedDeviceDiscovered()) {
                mDiscoveringDeviceIndex++;
            }
            if (mDiscoveringDeviceIndex < mDiscoveredTestbedDevicesAdapter.getCount() &&
                    mBluetoothLeService != null) {
                mBluetoothLeService.connect(mDiscoveredTestbedDevicesAdapter
                        .getTestbedDevice(mDiscoveringDeviceIndex), false);
                invalidateOptionsMenu();
            } else {
                discoveringDone();
            }
            break;
```

Výpis zdrojového kódu 6.3: Přepínač uvnitř přijímače asynchronních zpráv – 2. část

Případ `BluetoothLeService.ACTION_TESTBED_ID_DATA_AVAILABLE` nastává jakmile je hodnota charakteristiky vyčtena. Dané zařízení ze seznamu je doplněno o zjištěné ID a osazené sen-

zory. Rovněž je s databází porovnán stav tohoto zařízení a pomocí ikony prezentován v seznamu. Zařízení je poté označeno příznakem za objevené a je od něj odpojeno. Poslední případ v přijímači – `BluetoothLeService.ACTION_GATT_DISCONNECTED` – indikuje odpojení od zařízení. Pokud je poslední připojené zařízení objevené, je posunut ukazatel na další nalezené zařízení a zároveň je s tímto zařízením zahájeno navazování spojení. Pokud z jakéhokoliv důvodu objevování zařízení selže, je po jeho odpojení s ním znovu navázáno spojení, dokud není objevení kompletní. Pokud jsou všechna zařízení objevena, je zavolána metoda `discoveringDone()`, která odregistruje přijímač a odstraní neobjevená zařízení, která v seznamu mohla zůstat, například přerušením objevování ze strany uživatele. Výpis zdrojového kódu 6.3 s popsanými případy je uveden výše.

Po dokončení metody `discoveringDone()` se aktivita nachází ve stavu, kdy může být zahájeno nové vyhledávání, nebo připojení k jednomu z nalezených a objevených zařízení. Připojení k vybranému zařízení se provádí krátkým dotykem na vybrané zařízení v seznamu. Korektně se připojit je možné pouze k zařízení, které je shodné se záznamem k databázi. Pokud uživatel vybere zařízení, které shodu neprokazuje, je informován o nutnosti takové zařízení restartovat a poté znova spustit celý proces vyhledávání. U zařízení, které je s databází shodné, je uživateli zobrazeno dialogové okno informující jej o stavu připojení, objevení dostupných služeb a povolení příjímání notifikací. Posléze spuštěna aktivita DatabaseActivity a současná aktivita DiscoverDevicesActivity je ukončena.

Jak již bylo zmíněno výše, metoda `onResume()` může spustit také cílené skenovaní po posledním dostupném zařízení. Děje se tak v případě, že hodnota globální proměnné `mFirstActivityRun` je pravdivá. Poté je volána metoda `scanLastConnectedTestbedDevice()`.

```
try {
    mLastConnectedTestbedDevice = mTestbedDatabase.selectLastConnectedTestbedDevice();
    ScanFilter scanFilter = new ScanFilter.Builder()
            .setDeviceName(mLastConnectedTestbedDevice.getName())
            .setDeviceAddress(mLastConnectedTestbedDevice.getMacAddress())
            .build();
    mBluetoothLeScanner.startScan(Collections.singletonList(scanFilter),
            new ScanSettings.Builder().build(), mScanLastConnectedTestbedDeviceCallback);
    mScanningLastConnectedTestbedDeviceCountDownTimer.start();
    mScaningForLastConnectedTestbedDevice = true;
    mConnectingToLastConnectedTestbedDeviceDialog.show();
} catch (TestbedDatabase.EmptyCursorException e) {
    scanTestbedDevices(SCAN_COMMAND.START_SCANING);
}
```

Výpis zdrojového kódu 6.4: Sekvence zahájení skenování po posledním připojeném zařízení.

Tato metoda nejprve vybírá z databáze poslední připojené zařízení. Poté nastaví jeho název a MAC adresu jako vstupní filtry. Instanci Bluetooth skeneru jsou poté stejně jako v případě necíleného vyhledávání předány filtry a odkaz na callback. Skenování je odstartováno společně s časovačem, který po určité době (10 sekund) skenování zastaví. Současně je také uživateli zobrazeno

dialogové okno, které jej informuje o aktuálním stavu a průběhu skenování a připojování. Blok try – catch odchytává výjimku, která je instancí třídy `TestbedDatabase.EmptyCursorException`. Vyhození této výjimky systémem znamená, že požadovaný dotaz do databáze nevrátil žádný výsledek, což znamená, že žádné poslední připojené zařízení v databázi dosud neexistuje a není možné jej tedy vyhledat. Tato výjimka je ošetřena spuštěním skenování po všech dostupných zařízeních. Typickým výskytem této výjimky je první spuštění po instalaci aplikace, kdy nebylo zatím navázáno žádné spojení. Ukázka sekvence, která spouští cílené skenování je na výpisu zdrojového kódu 6.4. Instance s názvem `mScanLastConnectedTestbedDeviceCallback` je callback, který je volán v případě shody nalezeného zařízení se zadaným filtrem. Metoda callbacku `onScanResult()` volaná v případě nalezení zařízení, skenování okamžitě pozastavuje, deaktivuje časovač skenování a zahajuje připojování k nalezenému zařízení. O úspěšném nalezení zařízení je uživatel informován opět prostřednictvím dialogového okna.

O stavu procesu připojování informuje služba aktivitu opět prostřednictvím přijímače událostí. Instance přijímače `mConnectingToLastConnectedTestbedDeviceBroadcastReceiver`, který byl po nalezení posledního připojeného zařízení zaregistrován, zpracovává události během připojování k poslednímu zařízení. Události v těle přijímače jsou opět odchyceny pomocí přepínače. Ten reaguje na připojení a objeveni dostupných služeb, vyčteni ID zařízení a dostupných senzorů a povolení přijímaní notifikací. V případě úspěšného projití všemi sekvencemi je na konci procesu spuštěna aktivita DatabaseActivity a současná aktivita DiscoverDevicesActivity je ukončena.

Aktivita DiscoverDevicesActivity v metodě `onResume()` registruje instanci dalšího přijímače asynchronních událostí – `mSystemServicesStateChangedBroadcastReceiver`. Tento přijímač reaguje na vypnutí některého z potřebných adaptérů pro chod aktivity a zároveň navádí uživatele k řešení vzniklého problému.

6.2.3 Implementace aktivity DatabaseActivity

Tato aktivita hostuje trojici fragmentů, které se zobrazují dle volby uživatele ve vyhrazeném prostoru layoutu. V první volané metodě `onCreate()` je získána instance databáze a také zařízení testbed, které je předáno z předchozí aktivity DiscoverDevicesActivity. Dále jsou naplněny instance jednotlivých fragmentů a dalších tříd potřebných pro práci s nimi. Inicializována je také hodnota aktuálního intervalu a jeho úrovně potřebná pro třídění a prezentaci naměřených dat. Následně aktivita provede inicializaci horní navigace a akcí spojených s interakcí s touto navigací a patřičnou změnou fragmentu. Inicializována je také spodní navigace a akce, které reagují na změnu aktuálního senzoru voláním patřičných metod aktuálního fragmentu vybraného horní navigací. Rozsah spodní navigace je logicky závislý na osazených senzorech, a tak je například odebrána možnost zvolit senzor srdečního tepu, pokud jím není daný produkt testbed vybaven. Jako poslední je poté v této metodě inicializováno dialogové okno, které se uživateli zobrazí při výpadku Bluetooth spojení s produktem testbed.

Metoda `onResume()` volána po skončení metody `onCreate()` se připojuje ke službě BluetoothLeService běžící na pozadí a registruje přijímač asynchronních událostí, který uživatele provede k opětovnému navázaní ztraceného Bluetooth spojení. Zároveň volá dle právě aktivního fragmentu metodu, která aktualizuje prezentovaná data o nové hodnoty zatímco byla aplikace uživatelem skryta.

Před přechodem jiné aktivity nebo aplikace do popředí je v metodě `onPause()` aktivita odpojena od služby a zároveň je odregistrován přijímač asynchronních událostí pro navázaní ztraceného spojení.

```
@Override
public void onBackPressed() {
    AlertDialog closeAppliactionDialog = new AlertDialog.Builder(this)
            .setIcon(AppCompatResources.getDrawable(
                    this, R.drawable.ic_shutdown_20dp_color_vsb))
            .setTitle(getString(R.string.dialog_close_application_title))
            .setPositiveButton(getString(R.string.dialog_close_application_positive_button),
                    (dialog, which) -> finish())
            .setNeutralButton(getString(R.string.dialog_close_application_neutral_button),
                    (dialog, which) -> {
                        mDestroyBackgroundService = false;
                        mBluetoothLeService.disconnect();
                        finish();
                        startActivity(new Intent(
                                DatabaseActivity.this, DiscoverDevicesActivity.class)
                                .putExtra(StartUpActivity.FIRST_RUN_APPLICATION_KEY, false));
                    })
            .setNegativeButton(getString(R.string.dialog_close_application_negative_button),
                    null)
            .create();
    closeAppliactionDialog.show();
}
```

Výpis zdrojového kódu 6.5: Přetížení metody `onBackPressed()`

Tato aktivita také přetěžuje metodu `onBackPressed()`, která je standardně volána jako událost po stisknutí tlačítka zpět a bez předchozího varování ukončuje aktivitu. V případě této aktivity je uživateli po stisku tlačítka zobrazeno dialogové okno, které mu umožňuje své rozhodnutí aktivitu a tím i aplikaci ukončit ještě přehodnotit, a nabízí mu zůstat, nebo se jen odpojit od právě připojeného zařízení. Na výpisu zdrojového kódu 6.5 je zobrazeno přetížení této metody.

Metoda `exportData()` slouží k exportu dat do souboru ve formátu CSV. Na začátku metody je vytvořeno dialogové okno, které nabízí výběr typu exportovaných dat pomocí checkboxů a také seznam zařízení uložených v databázi, ze kterých mají být vybrané typy dat exportovány. Zbytek algoritmu pro export dat se odehrává v callbacku po stisku tlačítka pro export. Je iterováno napříč všemi vybranými zařízeními a z databáze jsou vybrány všechny záznamy shodné s aktuálním zaří-

zením a vybranými typy dat. Současně je také dle aktuálního časového razítka, vybraných zařízení a dat skládán budoucí název souboru. Na konci callbacku je vytvořen nový soubor a nad proměnou `mOnDataExportedActivityLanucher`, která je instancí třídy `ActivityResultLauncher<Intent>` je volána metoda `launch()`, jež spustí systémovou aktivitu pro zvolení cesty k uložení souboru. Po zvolení cesty souboru je volán callback `onActivityResult()`, který je součástí právě výše zmíněné instance. V jeho těle jsou pak procházeny jednotlivé zařízení a jim náležející záznamy jsou zapisovány do souboru. Po skončení je soubor uložen, zavřen a připraven k dalšímu použití.

Implementace fragmentu BaseVisualisationFragment

Fragment BaseVisualisationFragment je potomkem třídy Fragment a zároveň rodičem trojice fragmentů, které jsou hostovány aktivitou DatabaseActivity. Třída fragmentu je označena klíčovým slovem `abstract`, což znamená, že není přímo instancovatelná. Fragment také nemá vlastní frontendovou část.

Třída obsahuje několik globálních proměnných, které jsou sdílené napříč všemi třemi hlavními fragmenty. Jedná se zejména o textové pole a tlačítka.

Metoda `getIntervals()` vrací hranice intervalu jako pole instancí `Date` o délce 2. První prvek je počáteční časové razítko a druhý koncové časové razítko. Metoda vrací tyto intervaly na základě argumentů, kterými jsou datum požadovaného intervalu a úroveň. Pokud je jako vstup vloženo datum `12. 04. 2021 11:22:33` a úroveň rovna 12 (`Calendar.MINUTE`) tak metoda vrátí data: `12. 04. 2021 11:00:00` a `12. 04. 2021 11:59:59`. Tato dvojice časových razítek pak slouží pro dotaz k výběru záznamů z databáze.

Metody `dataAvailable()` a `dataLoading()` zviditelňují, případně skrývají prvky layoutu, během načítání, případně nedostupnosti dat.

Metoda `sortRecordsByInterval()` seřadí seznam dat vybraný z databáze do zadaných intervalů. Touto metodou je tak možné například seskupit data za vybraný měsíc do jednotlivých dní. Nejdříve je metodou vytvořen seznam nulových prvků s délkou, která odpovídá počtu prvků intervalu. Pokud budeme například řadit záznamy za den v měsíci podle hodin, je délka tohoto seznamu 24. Poté je iterováno napříč všemi záznamy a každý záznam je přiřazen do seznamu dle časového razítka.

```
protected List<Record> sortSteps(Record lastValueBeforeInterval, List<Record> records) {
    List<Record> sortedSteps = new ArrayList<>();
    float steps;
    Record lastRecord = lastValueBeforeInterval;
    Record newRecord;
    for (Record record : records) {
        if (lastRecord.getValue() < record.getValue()) {
            steps = record.getValue() - lastRecord.getValue();
        } else {
            steps = record.getValue();
```

```
            }
            lastRecord = record;
            newRecord = record.clone();
            newRecord.setValue(steps);
            sortedSteps.add(newRecord);
        }
        return sortedSteps;
    }
```

Výpis zdrojového kódu 6.6: Metoda `sortSteps()`

Metoda `sortSteps()` slouží ke zjištění celkového úhrnu kroků. Vstupními argumenty jsou záznamy za daný interval a počáteční záznam. Ten je zpravidla posledním záznamem, pořízeným před intervalem, ze kterého pochází vstupní záznamy. Metoda prochází všechny záznamy a porovnává je z předchozím záznamem. Pokud je hodnota předchozího záznamu menší než hodnota současného, je počet kroků roven rozdílu těchto hodnot. Pokud je ale předchozí záznam větší, je počet kroků roven nové hodnotě, protože došlo s největší pravděpodobností k restartování čítače kroků na produktu testbed. Počet kroků je poté přidán do seznamu, který je následně vrácen jako návratová hodnota metody. Metoda `sortSteps()` je na výpisu zdrojového kódu 6.6.

Implementace fragmentu OverviewFragment

Tento fragment je potomkem výše popsaného fragmentu BaseVisualisationFragment. Fragment poskytuje uživateli základní přehled o záznamech uložených v databázi a posledních pořízených hodnotách.

V metodě `onCreateView()`, která je volána vždy po přechodu fragmentu do popředí, jsou inicializována textová pole, tvořící jednotlivé displeje a navigace sloužící k přepínání mezi jednotlivými statistickými ukazateli.

Při změně senzoru volá aktivita DatabaseActivity metodu `sensorChnaged()`. Metoda na základě aktuálně zvoleného typu senzoru připraví navigaci se statistickými ukazateli, které jsou pro daný typ senzoru dostupné. Poté jsou volány metody `updateLastValue()` a `updateIntervalValue()`.

```
        case R.id.temperature:
            if (mDatabaseActivity.getBluetoothLeService() != null &&
                    mDatabaseActivity.getBluetoothLeService().getmLastTemperatureValue() != null)
                        {
                updateValueAndTimestampTextviewPair(mTxvLastValue, mTxvLastValueTimeStamp,
                        mDatabaseActivity.getBluetoothLeService().getmLastTemperatureValue(),
                        mDatabaseActivity.getBluetoothLeService().getmLastTemperatureTimeStamp(),
                        RecordValueFormatter.PATERN_3_2, blink);
            } else {
                mTestbedDatabase.selectFirstRecordLessThanTimeStamp(mTestbedDevice,
                        BluetoothLeService.TEMPERATURE_DATA, new Date(), databaseResult -> {
                            if (databaseResult instanceof DatabaseResult.Success) {
```

```
                    Record lastRecord = ((DatabaseResult.Success<Record>)
                        databaseResult).data;
                    updateValueAndTimestampTextviewPair(mTxvLastValue,
                            mTxvLastValueTimeStamp,
                            lastRecord.getValue(),
                            lastRecord.getTimeStamp(),
                            RecordValueFormatter.PATERN_3_2, blink);
                } else if (databaseResult instanceof DatabaseResult.Error) {
                    updateValueAndTimestampTextviewPair(mTxvLastValue,
                            mTxvLastValueTimeStamp, null, null,
                            null, blink);
                }
            });
    }
    break;
```

Výpis zdrojového kódu 6.7: Metoda `updateLastValue()`

Metoda `updateLastValue()` je volána při změně senzoru nebo při příjmu nové hodnoty z produktu testbed. Metoda nejdříve zjišťuje prostřednictvím hostované aktivity, jestli již služba BluetoothLeService obsahuje poslední známou hodnotu. Pokud ano, je tato hodnota včetně časového razítka prezentována uživateli v horním displeji. Pokud zatím služba žádnou poslední hodnotu neobsahuje (typicky po startu aplikace), vybere metoda poslední hodnotu z databáze. Ukázka aktualizace horního displeje o novou/poslední známou hodnotu teploty je na výpisu zdrojového kódu 6.7.

Metodou `updateIntervalValue()` je aktualizována jedna z trojice displejů sloužící k zobrazení významných statistických ukazatelů. Metoda, dle argumentu a aktuálního zvoleného senzoru pomocí metody `getStatisticalData()`, která vypočítá statistický ukazatel na základě vybraných záznamů z databáze, aktualizuje příslušný displej. Tedy hodnotu a časové razítko.

Implementace fragmentu ChartFragment

Dalším potomkem a zároveň jedním z trojice fragmentů je fragment ChartFragment. Tento fragment uživateli prezentuje jednotlivé záznamy v grafické podobě.

V metodě `onCreateView()` jsou inicializovány veškeré grafické prvky a callbacky, které jsou volány při interakci s nimi.

```
    case R.id.temperature:
        dataLoading(true);
        mTestbedDatabase.selectRecordsBetweenTimeStamp(
                mTestbedDevice,
                BluetoothLeService.TEMPERATURE_DATA,
                getIntervals(actualSortingInterval, actualSortingLevel),
                TestbedDatabase.SORT_BY.DEFAULT, TestbedDatabase.SORT_ORDER.DEFAULT,
                databaseResult -> {
                    if (databaseResult instanceof DatabaseResult.Success) {
```

```
                mRecords = ((DatabaseResult.Success<List<Record>>)
                    databaseResult).data;
                calendar.setTime(mTestbedDatabase.getFirstRecord(mRecords)
.getTimeStamp());
                years[0] = calendar.get(Calendar.YEAR);
                calendar.setTime(mTestbedDatabase.getLastRecord(mRecords)
.getTimeStamp());
                years[1] = calendar.get(Calendar.YEAR);

                setTemperatureData(getStatisticalDataByIntervals(
                        sortRecordsByIntervals(mRecords, actualSortingInterval,
                                actualSortingLevel)), years, animation);
                dataAvailable(true);
            } else if (databaseResult instanceof DatabaseResult.Error) {
                dataAvailable(false);
            }
        });
break;
```

Výpis zdrojového kódu 6.8: Metoda `updateChartData()`

Stěžejní metodou fragmentu je metoda `updateChartData()`. Po zavolání metody je zobrazen grafický indikátor, který indikuje načítání nových dat. Poté je na základě aktuálně zvoleného senzoru a aktuální úrovně časového intervalu vybrán set záznamů z databáze odpovídající zadaným kritériím. Záznamy jsou poté seřazeny do jednotlivých částí daného intervalu. Z těchto částí je pak spočítáno minimum, maximum, první a třetí kvartil, případně celkový úhrn. Tyto údaje pak slouží ke konstrukci jednotlivých záznamů v grafu. Po seřazení záznamu je dle typu dat volána jedna z metod `setPedometerData()`, `setHeartRateData()` nebo `setTemperatureData()`, jež se stará o samotné vykreslení grafu. Ukázka výběrů dat z databáze, jejich řazení a výpočet je na výpisu zdrojového kódu 6.8.

Výše popsaná trojice metod funguje na obdobném způsobu. Předané hodnoty za jednotlivé úseky časového intervalu jsou procházeny a z jejich hodnot je vytvořena instance třídy `BarEntry` nebo `CandleEntry`. Tyto instance jsou přidány do seznamu záznamů, který pak tvoří vodorovnou osu grafu. Při procházení je rovněž spočítáno globální minimu a maximum pro nastavení rozsahu svislé osy. Na konci každé z metod je volána metoda `renderChart()`. Tato metoda vykreslí graf na základě stavů checkboxů sloužící pro nastavení jednotlivých částí grafu.

Implementace fragmentu ListItemFragment

Posledním z trojice fragmentů hostovaných aktivitou DatabaseActivity je fragment ListItemFragment. Fragment prezentuje záznamy v podobě seznamu.

Úvodní metoda `onCreateView()` inicializuje grafické prvky, seznam a dialogové okno podrobnostmi o záznamu.

Aktualizace dat v seznamu je prováděna skrze metodu `updateListViewData()`, která je obdobou metody `updateChartData()` z fragmentu ChartFragment. Metoda dle zvoleného senzoru a aktuální úrovně časového intervalu vybere set záznamů z databáze odpovídající zadaným kritériím a zvolenému typu řazení. Záznamy jsou poté pomocí metody `setData()` přidány do seznamu.

Při stisku dvojice tlačítek ve spodní části fragmentu sloužících k řazení záznamu jsou volány metody `sortingByChanged()` a `sortingOrderChanged()`. Tyto metody přepíší text daného tlačítka, změní hodnotu globální proměnné a zavolají znovu metodu `updateListViewData()`, která vystaví nově seřazená data do seznamu.

6.2.4 Implementace služby BluetoothLeService

Na pozadí celé aplikace běží služba s názvem BluetoothLeService. Úkolem služby, jak již bylo mnohokrát dříve zmíněno, je navazování a udržování Bluetooth spojení s produktem testbed. Služba dále přijímá notifikace zaslané produktem a obsažená data ukládá do databáze. Služba také zajišťuje aktualizaci oznámení v horním stavovém řádku.

Služba je poprvé spuštěna na konci aktivity StartUpActivity. Při spuštění služby je volána metoda `onCreate()`. V této službě je získána instance lokální databáze, která je dále využívána napříč celou službou.

Při připojení aktivity (DiscoverDevicesActivity nebo DatabaseActivity) ke službě je volána metoda `onBind()`. Tato metoda vrací instanci třídy `LocalBinder`, která slouží pro následné vytvoření instance služby v připojené aktivitě.

Přípojení k Bluetooth zařízení se provádí pomocí metody `connect()`, odpojení pak pomocí metody `disconnect()`. Před samotným připojením produktu testbed je zaregistrovaná instance třídy `BluetoothGattCallback`. Tento callback zachycuje asynchronní události, které jsou volány během připojování a odpojování od Bluetooth zařízení.

```
@Override
public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
    if (newState == BluetoothProfile.STATE_CONNECTED) {
        if (mShowNotification) {
            mNotificationManager = NotificationManagerCompat.from(getApplicationContext());
            mNotificationManager.notify(NOTIFICATION_ID, Objects.requireNonNull(
                    buildNotification(getColor(R.color.colorPrimary)))
                    .setSmallIcon((R.drawable.ic_device_id_20dp_color_fei)).build());
        }
        if (mAutoReconnect) {
            mBluetoothGatt.discoverServices();
        }
        broadcastUpdate(ACTION_GATT_CONNECTED);
    } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
        if (mNotificationManager != null) {
            mNotificationManager.cancel(NOTIFICATION_ID);
```

```
            }
            if (mAutoReconnect) {
                connect(mTestbedDevice, true);
            }
            broadcastUpdate(ACTION_GATT_DISCONNECTED);
        }
    }
```

Výpis zdrojového kódu 6.9: Metoda `onConnectionStateChange()`

Při připojení nebo odpojení je volána metoda `onConnectionStateChange()`. V těle této metody je zjištěno, jaký stav nově nastal. Dle okolností je zobrazeno případně skryto upozornění v horním stavovém řádku a povoleno, případně zakázáno automatické znovu navázání ztraceného spojení (při výpadku). Pomocí metody `broadcastUpdate()` je zaslána událost o novém stavu do připojené aktivity. Pokud je povoleno automatické navazování spojení, je spuštěno objevování služeb. Metoda `onConnectionStateChange()` je zobrazena na výpisu zdrojové kódu 6.9.

Další z metod tohoto callbacku je metoda `onServicesDiscovered()`. Tato metoda je volána po objevení dostupných služeb. V těle metody jsou procházeny dostupné služby dokud není nalezena servisní služba. Hodnota charakteristiky je vyčtena a dle její hodnoty (osazených senzorů) jsou určeny charakteristiky ostatních služeb, u kterých budou povoleny notifikace. Do připojené aktivity je opět pomocí metody `broadcastUpdate()` zaslána událost o objevených službách. Zároveň je zapsána hodnota deskriptoru na první charakteristiku.

Metoda `onDescriptorWrite()` je volána poté, co je do deskriptoru konkrétní charakteristiky zapsána nová hodnota. Následně je ze seznamu charakteristik, kde mají být povoleny notifikace vybrána další, do jejíhož deskriptoru bude zapsána nová hodnota. Pokud je celý seznam zapsán, metoda posílá do připojené aktivity událost s informací, že všechny notifikace na požadovaných charakteristikách byly povoleny.

Metody `onCharacteristicRead()` a `onCharacteristicChanged()` jsou volány při vyčtení hodnoty nebo obdržení hodnoty pomocí notifikace. V metodách je zjištěno o jakou charakteristiku se jedná a na základě toho je tato hodnota zaslána do připojené aktivity.

Metoda `broadcastUpdate()` slouží jako prostředník pro zasílání asynchronních událostí do připojené aktivity. Jakmile je tato metoda zavolána v obslužném kódu služby, je na základě akce předané v argumentu metody vykonán daný případ.

```
    } else if
        (SampleGattAttributes.HEART_RATE_CHARACTERISTIC_UUID.equals(characteristic.getUuid())) {
        if (Integer.parseInt(characteristic.getStringValue(0)) != 0) {
            mTestbedDatabase.insertRecord(mTestbedDevice, BluetoothLeService.HEART_RATE_DATA,
                    Integer.parseInt(characteristic.getStringValue(0)), databaseResult -> {
                float heartRate = ((DatabaseResult.Success<Float>) databaseResult).data;
                intent.putExtra(HEART_RATE_DATA, heartRate);
                mLastHeartRateValue = heartRate;
                mLastHeartRateTimeStamp = new Date();
```

```
                    mNotificationManager.notify(NOTIFICATION_ID, Objects.requireNonNull(
                            buildNotification(getColor(R.color.colorPrimary)))
                            .setSmallIcon((R.drawable.ic_heart_20dp_color_fei)).build());
                    sendBroadcast(intent);
                });
        }
```

Výpis zdrojového kódu 6.10: Případ metody `broadcastUpdate()` po obdržení nové hodnoty srdečního tepu

Nejprve je z textové hodnoty, kterou nese charakteristika vyparsována celočíselná nebo reálná hodnota. Tato hodnota je poté vložena do databáze jako nový záznam. Jakmile je zápis do databáze dokončen, je nová hodnota aktualizovaná v notifikační liště a pomocí systémové metody `sendBroadcast()` odeslána zpráva. Na výpisu zdrojového kódu 6.10 je ukázka z metody `broadcastUpdate()`.

6.2.5 Implementace tříd TestbedDatabase a TestbedDatabaseHelper

Z hlediska struktury aplikace jsou tyto třídy významné pro správný chod aplikace.

Instance třídy `TestbedDatabase` se vyskytují napříč aktivitami a službou, které skrze ní komunikují s lokální databází. Třída obsahuje metody, které na základě vstupních argumentů formují požadovaný SQL dotaz, který je následně spuštěn nad databází. Třída také obsahuje metody `getRecordFromCursor()` a `getRecordsFromCursor()` pro získání jednotlivých záznamů a metody `getTestbedDeviceFromCursor()` a `getTestbedDevicesFromCursor()`, které získávají z výsledků jednotlivá zařízení. Metody procházejí instanci typu `Cursor`, která obsahuje řádky tabulky ovlivněné SQL dotazem. Metody vrací instance, případně seznamy instancí tříd `Record` a `TestbedDevice`. V případě, že vstupní argument nenese žádné ovlivněné řádky, je vyhozená vyjímka, která je instancí třídy `EmptyCursorException`.

Třída `TestbedDatabaseHelper` je potomkem třídy `SQLiteOpenHelper` sloužící k přímému ovládání lokální databáze. Třída obsahuje statické SQL dotazy sloužící k vytváření a mazaní tabulek. Dále jsou v této třídě využity statické třídy `Data` a `Device`, které implementují rozhraní `BaseColumns`. Tyto třídy slouží pro práci s jednotlivými sloupci.

```
public static TestbedDatabaseHelper getInstance(Context context) {
    if (mTestbedDatabaseHelper == null) {
        mTestbedDatabaseHelper = new TestbedDatabaseHelper(context);
    }
    return mTestbedDatabaseHelper;
}

private TestbedDatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Výpis zdrojového kódu 6.11: Implementace návrhového vzoru singleton ve třídě `TestbedDatabaseHelper`

Aby bylo zabráněno vytváření nekonečného množství instancí této třídy, což by mělo negativní vliv na výkon aplikace, bylo použito návrhového vzoru singleton. Tento návrhový vzor počítá pouze s jednou existující instancí třídy v jeden časový okamžik. Konstruktor třídy je proti neřízenému vytváření instancí chráněn modifikátorem typu `private`. Tento modifikátor omezuje volání konstruktoru pouze ze samotné třídy. K vytvoření instance slouží statická metoda `getInstance()`. Po jejím zavolání je zjištěno, zda-li je proměnná `mTestbedDatabaseHeleper` inicializovaná. Pokud není, je vytvořena instance pomocí privátního konstruktoru. Pokud je proměnná již inicializovaná, znamená to, že instance již existuje a může být předána jako návratová hodnota metody `getInstance()`. Na výpisu zdrojového kódu 6.11 je ukázka implementace tohoto návrhového vzoru ve třídě `TestbedDatabaseHelper`.

Kapitola 7

Testování

Poslední a také průběžnou fází vývoje a implementace navrhované mobilní aplikace bylo testování. Testování probíhalo jednak průběžně během vývoje v rámci ladění aplikace na reálném mobilním zařízení, a také v rámci nasazení aplikace v produkčním prostřední na vybrané skupině alfa testerů.

7.1 Testování a měření výkonu aplikace v průběhu vývoje

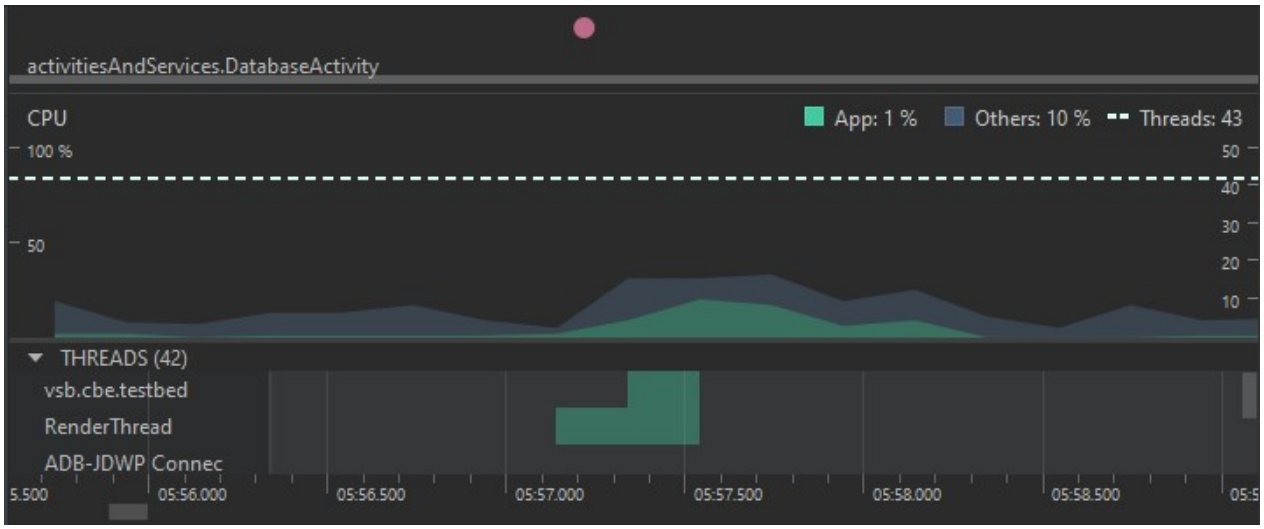
Použité vývojové prostředí, Android Studio, umožňuje kromě kompilace a spuštění vyvíjené aplikace na emulovaném nebo reálném zařízení také ladění a měření výkonu. Použití emulovaného zařízení je v tomto konkretním případě obtížné, jelikož emulování Bluetooth spojení je nestabilní a náročné. Pro průběžné testování tak byla využita dvojice mobilních zařízení výrobce **Xiaomi**. První je šesti a půl palcový smartphone **Xiaomi Redmi Note 8 Pro**. Zařízení disponuje osmi jádrovým procesorem o frekvenci 2,05 GHz a 6 GB operační pamětí. Zařízení běží na operačním systému Android verze 10. Druhé, starší, pěti palcové zařízení **Xiaomi Redmi 3S** s osmi jádrovým procesorem o frekvenci 1,4 GHz a 3 GB operační paměti pak běží na Androidu verze 6.

Pro ladění aplikace je nutné aktivovat ladicí nástroj. Ten na úkor sledování aplikace v reálném čase značně aplikaci zpomalí. Aktivaci je možné provést přímo při spuštění aplikace nebo v případě, že je nutné ladit některou ze vzdálenějších částí aplikace, lze ladění spustit až za běhu. Ladící nástroj se připojuje k mobilnímu zařízení pomocí soketu. Fyzická část spojení je realizovaná skrze rozhraní USB.

Pro zachycení konkrétního stavu laděné aplikace slouží breakpointy. Ty se umístí na řádek v kódu, kde se dle předpokladu nachází problematické místo. Při dosažení breakpointu při vykonávání programu se aktuální vlákno aplikace zastaví, avšak ostatní vlákna pracují dále. Po zastavení aplikace je uživateli k dispozici obsah ovlivněných proměnných. Pozastavené vlákno lze znovu spustit, přičemž běží až k dalšímu breakpointu, nebo je možné se zanořit hlouběji do volaných metod. Tímto způsobem je možné detailně zkoumat části zdrojového kódu a nalézt případnou implementační chybu.

Během vývoje byl také průběžně sledován vliv aplikace na využívání prostředků. Sledováno bylo zejména využití baterie, CPU a operační paměti. Sledování těchto údajů umožňuje integrovaný nástroj Android Studia, který se nazývá **Profiler**.

Během testování bylo zjištěno, že největší spotřeba energie nastává v momentě, kdy aplikace vyhledává dostupná Bluetooth zařízení. Z toho důvodů je vyhledání omezeno pouze na 10 sekund. Tato doba je dostatečně dlouhá na to, aby byla všechna Bluetooth zařízení v dosahu mobilního zařízení nalezena. U staršího zařízení není možné využití baterie pomocí tohoto nástroje určit, jelikož sledování energie je možné až od verze operačního systému 8 a výše.



Obrázek 7.1: Výsledek měření využití CPU při databázovém dotazu, zdroj: autor

Během průběžného testování bylo zjištěno, že SQL dotaz na výběr záznamů z databáze vytěžuje aplikaci natolik, že je to znát na její odezvě. Proto bylo přistoupeno k implementaci SQL dotazu do asynchronního vlákna, čímž nadále nedocházelo k zamrznutí aplikace při uživatelské interakci. Na obrázku 7.1 je záznam z nástroje **Profiler**, kde je měřeno využití CPU. V časovém okamžiku spuštění SQL dotazu, který je na obrázku indikován růžovým puntíkem, je patrné zvýšené využití jinak klidě pracujícího procesoru. Po necelé jedné sekundě zvýšené zátěže je výkon jednotky CPU opět v rozmezí několika málo procent. Pomineme-li stavy, kdy je spuštěna aplikace, aktivita nebo služba, tak k žádným vetším výkyvům vytížení CPU srovnatelných s tímto během běhu aplikace nedochází. Vytížení operační paměti zařízení je konstantní a ani složitější výpočty, jako například řazení záznamu nezpůsobí znatelný výkyv. Výše uvedená změna zatížení je srovnatelně stejná pro obě testovaná zařízení. Ačkoliv minimální verze API mobilního zařízení musí být 23 (Android 6) a vyšší běží aplikace lépe a plynuleji na novějších a výkonnějších zařízeních s posledními verzemi operačního systému.

7.2 Jednotkové a instrumentační testování

Aplikace pro operační systém Android může běžet na různých zařízení různých výrobců, kteří navíc mohou funkcionálně svou verzi systému modifikovat. Modifikace nezřídka způsobí zásadní změnu chování systému, která může mít samozřejmě vliv také na navrhovanou aplikaci. Neočekávanému chování aplikace a jejímu případnému pádu lze předcházet testy softwaru. Základními testy jsou jednotkový test (unit test) a instrumentační test. Jednotkovým testem se provádí funkční testování třídy a jejich metod, které není závislé na prostředí. To zanmená, že jednotkové testy nevyužívají kontext aplikace, připojení k sítí a přístup k úložišti. Pro testování částí aplikace, které některý z výše vypsaných přístupů vyžadují se používá instrumentační (funkční) test.

7.2.1 Jednotkový test

Za jednotkový test je považován test, který testuje pouze danou konkrétní jednotku. Jednotkový test by měl být v ideálním případě nezávislý na ostatních a izolovaný od ostatních částí softwaru. Android studio umožňuje přímou implementaci knihovny **JUnit**, která slouží pro psaní a spouštění testů. Jednotkové testy jsou uloženy jako sety ve vlastní třídě, která se nachází v kopii produkčního balíčku. Testy, které píšeme tak nejsou součástí produkční verze aplikace.

V případě navrhované aplikace byl jednotkový test využit pro testování třídy `StatisticalData` a jejich metod. Tato třída na základě setu záznamů spočte statistické ukazatele. Set testů pro testovanou třídu se nachází v samostatné třídě `StatisticalDataUnitTest`.

```
private static final String DATA_KEY = BluetoothLeService.TEMPERATURE_DATA;
// "cz.vsb.cbe.testbed.TEMPERATURE_DATA"

private StatisticalData mStatisticalData;

@Before
public void setUp() {
    List<Record> testRecords = new ArrayList<>();
    testRecords.add(new Record(1234, 555, DATA_KEY, -4.24f, 1617524535000L));
    testRecords.add(new Record(1236, 555, DATA_KEY, 18.25f, 1617524536000L));
    testRecords.add(new Record(1237, 555, DATA_KEY, 20.12f, 1617524537000L));
    testRecords.add(new Record(1238, 555, DATA_KEY, 21.34f, 1617524538000L));
    testRecords.add(new Record(1242, 555, DATA_KEY, 22.55f, 1617524539000L));
    testRecords.add(new Record(1245, 555, DATA_KEY, 23.78f, 1617524540000L));
    testRecords.add(new Record(1246, 555, DATA_KEY, 24.98f, 1617524541000L));
    testRecords.add(new Record(1247, 555, DATA_KEY, 27.45f, 1617524542000L));
    testRecords.add(new Record(1248, 555, DATA_KEY, 36.54f, 1617524543000L));
    mStatisticalData = new StatisticalData(testRecords);
}
```

Výpis zdrojového kódu 7.1: Metoda `setUp()` ve třídě jednotkového testu `StatisticalDataUnitTest`

Třída obsahuje metodu `setUp()`, která je anotována značkou `@Before`. To znamená, že je metoda volána před každým zahájením testu pro inicializaci vstupních parametrů jednotkového testu. V tomto případě, je vytvořen seznam, který je posléze naplněn umělými daty pro ověření jednotlivých metod testované třídy. Seznam obsahuje devět po sobě časově jdoucích záznamů stejného typu dat od jednoho zařízení. Hodnoty jsou záměrně seřazeny vzestupně, ale na funkci testu, nebo testované metody to nemá žádný vliv. Po přidaní každého záznamu do seznamu je vytvořena instance testované třídy. Metoda `setUp()` je uvedena na výpisu zdrojového kódu 7.1.

```
@Test
public void testGetDataSetSize_isCorrect() {
    assertEquals(9, mStatisticalData.getDataSetSize());
}

@Test
public void testGetDataSetSum_isCorrect() {
    assertEquals(190.77f, mStatisticalData.getDataSetSum(), 0.01f);
}

@Test
public void testGetMeanValue_isCorrect() {
    assertEquals(21.19, mStatisticalData.getMeanValue(), 0.01f);
}

@Test
public void testGetMinValue_isCorrect() {
    assertEquals(-4.24f, mStatisticalData.getMinValue().getValue(), 0f);
}
```

Výpis zdrojového kódu 7.2: Metody jednotkových testů ve třídě `StatisticalDataUnitTest`

Metoda, která obsahuje samotný test je uvozena anotací `@Test`. Každá z testových metod volá metodu `assertEquals()`, která ověřuje danou funkcionalitu a produkuje výsledek testu. Testová metoda `testGetDataSetSize_isCorrect()` testuje metodu třídy `StatisticalData` s názvem `getDataSetSize()`. Jak je již z názvu patrné, metoda vrací hodnotu rovnou velikosti setu, ze kterého jsou statistické ukazatelé počítány. Při psaní testu víme, že testujeme na seznamu devíti umělých záznamů. Proto je číslo 9 předáno jako první argument metody `assertEquals()`. Jako druhý argument je předána právě testovaná metoda `getDataSetSize()`. V případě, že je tato metoda správně implementovaná a vrátí předpokládaný výsledek, je výsledek testu úspěšný, v opačném případě je neúspěšný s možností nahlédnutí na skutečný výsledek. Obdobně funguje také testová metoda `testGetMeanValue_isCorrect()`, která testuje výpočet průměrné hodnoty. V případě tohoto testu je metodě `assertEquals()` předán navíc argument, který stanovuje povolenou odchylku při výpočtu pro uznaní testu jako úspěšného. Set jednotkových testů je uveden na výpisu zdrojového kódu 7.2.

7.2.2 Instrumentační test

Za instrumentační test se považuje test části aplikace, která využívá některý z externích prostředků. Tím se tento test odlišuje od jednotkového testu, protože již není a nemůže být nezávislý a izolovaný od ostatních části softwaru, protože tyto části cíleně pro svůj běh využívá. Pro psaní a běh testů se využívá knihovna **AndroidJUnitRunner**. Samotný test neběží jako v případě jednotkového testu na lokálním Java engine, ale na fyzickém, nebo emulovaném Android zařízení. Výsledek testu je tak závislý jednak na implementaci testované metody, a také na použitém zařízení a jeho datech.

V navrhované aplikaci bylo instrumentačních testů využito při testování třídy `TestbedDatabase` a jejích metod pro ověření výsledků, které jsou vráceny na základě SQL dotazu. Set testů pro tuto třídu je umístěn ve třídě s názvem `TestbedDatabaseInstrumentationTest`.

```
@Before
public void setUp() {
    Context context = InstrumentationRegistry.getInstrumentation().getTargetContext();
    mTestbedDatabase = new TestbedDatabase(context);
    mTestbedDevice1 = new TestbedDevice(8228, "Testbed", 7, "88:6B:0F:D3:0E:CF", 0,
        1614943832203L);
    mTestbedDevice2 = new TestbedDevice(8228, "Testbed", 4, "88:6B:0F:D3:0E:CF", 1,
        1614943832203L);
    mTestbedDevice3 = new TestbedDevice(8228, "Testbed", 7, "88:6B:0F:D3:0E:CE", 0,
        1614943832203L);
    mTestbedDevice4 = new TestbedDevice(1234, "Testbed v2", 4, "AA:BB:CC:DD:EE:FF", 0,
        1234567890000L);
}
```

Výpis zdrojového kódu 7.3: Metoda `setUp()` ve třídě integračního testu `TestbedDatabaseInstrumentationTest`

Třída obsahuje metodu `setUp()` anotovanou značkou `@Before`, která je stejně jako v případě jednotkového testu volána před každým zahájením testu pro inicializaci vstupních parametrů. V metodě je vytvořena instance testované třídy pro připojení k databázi a následně inicializována čtveřice produktů testbed, na kterých bude testována metoda pro zjištění shody se záznamem v databázi, která se využívá pro ověření nalezených zařízení. První zařízení je přesná kopie zařízení, které je uložené v databázi. Druhé zařízení má jiný údaj o osazených senzorech (3. argument), než záznam v databázi. Tato odchylka může reálně nastat například při špatné inicializaci produktu testbed. Třetí zařízení má pak modifikovanou MAC adresu (4. argument). Toto zařízení pak simuluje například úpravu FW Bluetooth modulu, kdy nechtěně došlo ke změně MAC adresy. Čtvrté zařízení je pak zcela uměle vytvořené neznámé zařízení. Inicializace zařízení je uvedena na výpisu zdrojového kódu 7.3

```
@Test
public void testTestbedDevice1_STORED_CONSISTENTLY() {
    long result = mTestbedDatabase.getStoredStatusOfTestbedDevice(mTestbedDevice1);
```

```
    assertEquals((long) TestbedDevice.STORED_CONSISTENTLY, result);
}

@Test
public void testTestbedDevice2_STORED_BUT_MODIFIED() {
    long result = mTestbedDatabase.getStoredStatusOfTestbedDevice(mTestbedDevice2);
    assertEquals((long) TestbedDevice.STORED_BUT_MODIFIED, result);
}

@Test
public void testTestbedDevice3_STORED_BUT_MODIFIED() {
    long result = mTestbedDatabase.getStoredStatusOfTestbedDevice(mTestbedDevice3);
    assertEquals((long) TestbedDevice.STORED_BUT_MODIFIED, result);
}

@Test
public void testTestbedDevice4_NOT_STORED() {
    long result = mTestbedDatabase.getStoredStatusOfTestbedDevice(mTestbedDevice4);
    assertEquals((long) TestbedDevice.NOT_STORED, result);
}
```

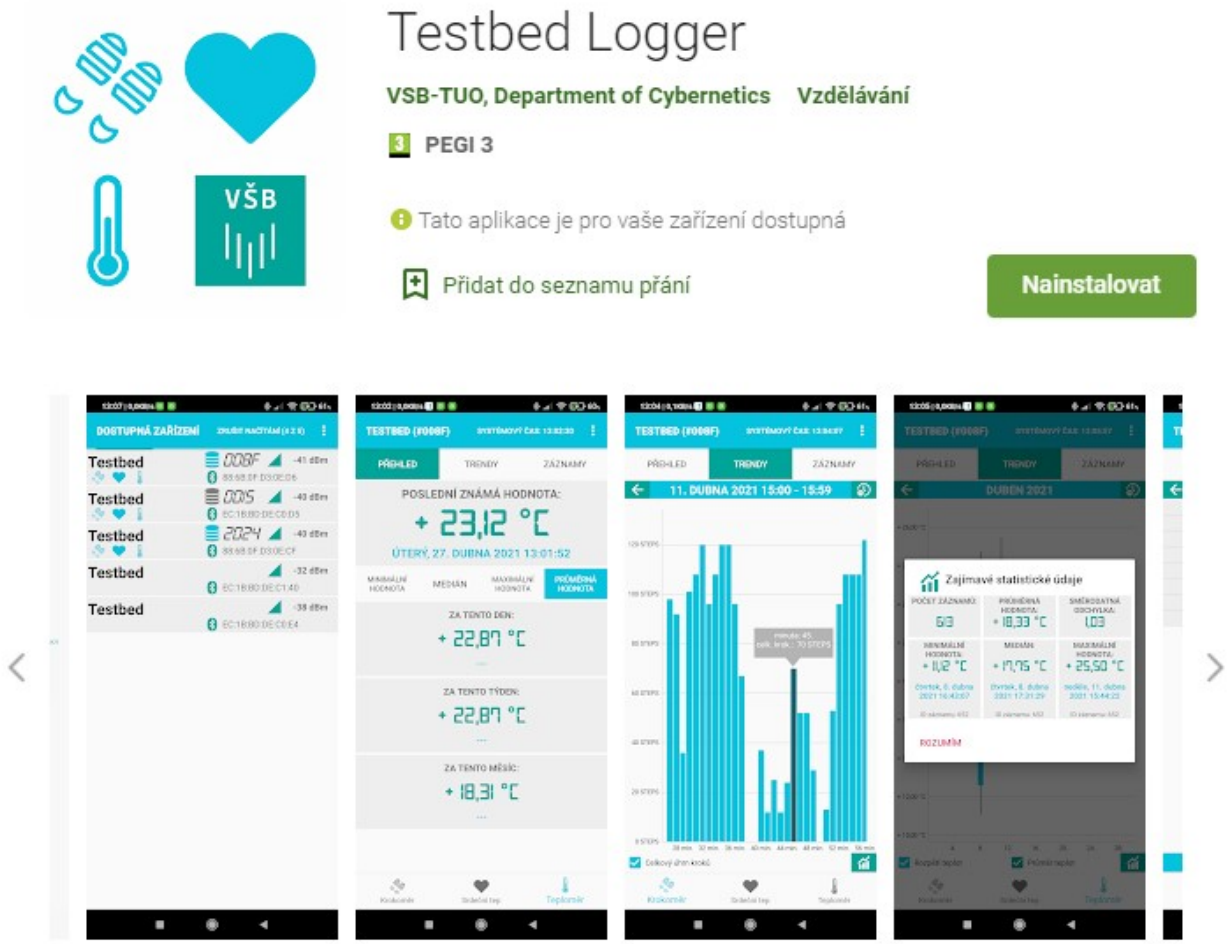
Výpis zdrojového kódu 7.4: Metody instrumentačních testů ve třídě `TestbedDatabaseInstrumentationTest`

Testovací metody jsou opět uvozeny anotací `@Test`. První testovací metoda této třídy s názvem `testTestbedDevice1_STORED_CONSISTENTLY()` ověřuje, zda-li je návratová hodnota testované metody `getStoredStatusOfTestbedDevice()` shodná s `TestbedDevice.STORED_CONSISTENTLY`. Jako předpoklad pro úspěšný výsledek testu je do testované metody vloženo právě první zařízení, které je kopií zařízení uloženého v databázi. Metody `testTestbedDevice2_STORED_BUT_MODIFIED()` a `testTestbedDevice3_STORED_BUT_MODIFIED()` jsou dalšími testovacími metodami, které testují metodu `getStoredStatusOfTestbedDevice()` na shodu s `TestbedDevice.STORED_BUT_MODIFIED`. Jako vstupní argument je do testované metody předána dvojice modifikovaných zařízeních. Poslední testovací metoda, `testTestbedDevice4_NOT_STORED()` pak testuje výsledek testované metody `getStoredStatusOfTestbedDevice()` se zcela neznámým zařízením na shodu s hodnotou `TestbedDevice.NOT_STORED`.

Set instrumentačních testů je uveden na výpisu zdrojového kódu 7.4.

7.3 Publikace aplikace v obchodě Google Play a alfa testování

Poslední částí práce bylo testování aplikace na uzavřené skupině alfa testerů.



Obrázek 7.2: Snímek obrazovky se záznamem v obchodě Google Play, zdroj: autor

Samotnému procesu alfa testování předcházela publikace aplikace v obchodě **Google Play**. Jedná se o známý repositář ověřených aplikací, který spravuje společnost Google. Publikování v obchodě je podmíněno zaplacením jednorázového registračního poplatku pro vývojáře, jehož výše je době psaní této práce $25 USD. Poté může vývojář mobilních aplikací využívat všechny funkce, které mu konzole nabízí. Skrze konzoli se spravuje název, grafika a popisy aplikace. Dále zde vývojář

nahrává jednotlivá vydání (verze) aplikace a také skrze konzoli může inkasovat finanční odměnu, kterou vybírá za používání nabízené aplikace. U každého nového vydání aplikace probíhá důkladná kontrola ze strany provozovatele obchodu. Zejména je kontrolováno, zda obsah aplikace souhlasí s deklarovaným popisem a věkovým určením aplikace. Vývojář musí také vystavit prohlášení o ochraně osobních údajů. Na obrázku 7.2 je snímek obrazovky z obchodu Google Play.

Pro publikování aplikace prostřednictvím konzole je nutné vygenerovat podepsaný balíček **APK**, případně novější a více kompresní variantu nazývanou jako **Andorid App Bundle**. Pro generování obou typu balíčků, je nutné vytvořit privátní klíč, kterým balíček podepíšeme. Ověření klíče na straně obchodu, kde je posléze balíček jako nové vydání nahrán funguje na principu asymetrické kryptografie.

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"
    defaultConfig {
        applicationId "cz.vsb.cbe.testbed"
        minSdkVersion 23
        targetSdkVersion 30
        versionCode 5
        versionName "1.11"
        testInstrumentationRunner "androidx.graph_navigation.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
                'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

Výpis zdrojového kódu 7.5: Část souboru `build.gradle`

Parametry aktuálního vydání jsou definovány v souboru `build.gradle`. Během publikování nové verze bychom neměli měnit název balíčku. Po aktualizaci by se aplikace nainstalovala jako nová a nebyla by nahrazena starší verzí. Zároveň hodnota parametru `versionCode` musí být s každým novým publikování zvýšena minimálně o jedna. Parametrem `versionName` pak specifikujeme textovou hodnotu verze aplikace, která je zobrazena také u záznamu v obchodě Google Play. Na výpisu zdrojového kódu 7.5 je uvedena část ze souboru `build.gradle`

Poté, co byla aplikace úspěšně publikována v obchodě Google Play byl vybrán okruh uživatelů, kteří budou aplikaci v produkčním nasazení testovat. Alfa-testerům byly rozdány produkty testbed ve finální podobě, aby pomocí nich vyzkoušeli funkcionalitu aplikace na různých typech zařízení. Pro alfa testování byla dostupná následující zařízení:

- **Xiaomi Redmi Note 8 Pro**
- **Xiaomi Redmi 3S**
- **Xiaomi Redmi Note 8**
- **Huawei Mate 20 Pro**
- **Samsung Galaxy M31s M317F**
- **Xiaomi Mi 9**

Dle zpětných vazeb uživatelů běžela aplikace bez problému a svižně. Nutno podotknout, že všechna zařízení vyjma **Xiaomi Redmi 3S** nejsou starší více než dva roky a běží na operačním systému Android verze 9.0 a vyšší.

Pouze u zařízení Xiaomi Redmi Note 8 se během produkčního testování vyskytl problém, kdy po objevení dostupných služeb u Bluetooth zařízení nebylo možné tyto služby vyčíst a pracovat s nimi. Dle reportingu podobného problému na jiných zařízeních bylo přistoupeno k implementaci zpoždění mezi obdržením události o objevených službách a zavoláním metody pro jejich vyčtení. Toto zpoždění bylo realizováno uspáním vlákna po dobu 200 ms.

Závěr

Cílem práce bylo navrhnout, implementovat a otestovat aplikaci pro mobilní operační systém Android, která bude prostřednictvím technologie Bluetooth komunikovat s produktem testbed, shromažďovat jím naměřená data a prezentovat je vhodnou formou uživateli mobilní aplikace.

První část práce shrnuje obecně známé teoretické poznatky, které se týkají technologií souvisejících se čtvrtou průmyslovou revolucí a pojmu testbed. Dále jsou shrnuty poznatky týkající se používaných mobilních platforem, členění architektury aplikace v mobilním operačním systému Android a technologie Bluetooth 4.0 Low Energy.

V druhé části práce je pak popsán samotný návrh aplikace prostřednictvím UML modelování. V rámci návrhu aplikace byly řešeny jednotlivé případy užití ze strany uživatele, možné scénáře a struktura lokální databáze. Návrh aplikace plynule přešel v nativní implementaci aplikace pro operační systém Android. Nejprve byly navrženy prvky UI, které byly posléze oživeny implementací zdrojového kódu v objektově orientovaném programovacím jazyce Java. Během implementace byla funkčnost jednotlivých částí zdrojového kódu ověřována pomocí jednotkových a instrumentačních testů. Navržená aplikace byla publikována v obchodě aplikací Google Play a podrobena produkčnímu testování na vybrané skupině uživatelů.

V rámci práce se povedlo navrhnout a implementovat mobilní aplikaci, která dokáže skrze Bluetooth rozhraní komunikovat s produktem testbed. Aplikace zvládá spravovat známá zařízení, ukládat a zpětně vyčítat získané záznamy a exportovat je do strojově čitelného formátu. Implementovaná aplikace byla podrobena ladění výkonu, jednotkovému a instrumentačnímu testovaní a v neposlední řadě také produkčnímu testování na skupině alfa testerů.

Aplikace je volně stažitelná a připravena stát se součástí testbedu CPIT TL3 Vysoké školy báňské – Technické univerzity v Ostravě. Budoucí rozšíření diplomové práce se může ubírat návrhem a implementací multiplatformní aplikace, jejíž použití nebude závisle na operačním systému.

Literatura

1. CEJNAROVÁ, Andrea. *Od 1. průmyslové revoluce ke 4.* [Online]. Praha: Business Media CZ s. r. o., 2014 [cit. 2020-12-18]. Dostupné z: `https://www.technickytydenik.cz/rubriky/ekonomika-byznys/od-1-prumyslove-revoluce-ke-4_31001.html`.
2. KORBEL, Petr. *Průmyslová revoluce 4.0: Za 10 let se továrny budou řídit samy a produktivita vzroste o třetinu* [online]. Praha: Economia, a.s., 2007 [cit. 2020-12-18]. Dostupné z: `https://byznys.ihned.cz/c1-64009970-prumyslova-revoluce-4-0-za-10-let-se-tovarny-budou-ridit-samy-a-produktivita-vzroste-o-tretinu`.
3. LASI, Heiner; FETTKE, Peter; KEMPER, Hans-Georg; FELD, Thomas; HOFFMANN, Michael. Industry 4.0. *Business & Information Systems Engineering*. 2014-08, roč. 6, č. 4, s. 239–242. ISSN 1867-0202.
4. DOSTÁL, Jiří. Průmysl 4.0 a Společnost 5.0 – výzvy pro změnu (nejen) technického vzdělávání. *Technika a vzdelávanie*. 2017-01, roč. 6, s. 49–54.
5. MUDLE, Kent. *Home Smart IoT Home: Domesticating the Internet of Things* [online]. New York: Toptal [cit. 2020-12-18]. Dostupné z: `https://www.toptal.com/designers/interactive/smart-home-domestic-internet-of-things`.
6. KOĎOUSKOVÁ, Barbora. *INTERNET VĚCÍ (IOT): DEFINICE, PŘÍKLADY VYUŽITÍ, PRODUKTY* [online]. Praha: Rascasone s.r.o., 2020 [cit. 2020-12-18]. Dostupné z: `https://www.rascasone.com/cs/blog/iot-internet-veci-definice-produkty-historie`.
7. ELKHODR, Mahmoud; SHAHRESTANI, Seyed A.; CHEUNG, Hon. Emerging Wireless Technologies in the Internet of Things: a Comparative Study. *CoRR*. 2016, roč. abs/1611.00861. Dostupné z arXiv: `1611.00861`.
8. MILAN, Ing. Klauz. *Bezdrátové technologie pro internet věcí* [online]. Liberec: CADware s.r.o., 2018 [cit. 2020-12-18]. Dostupné z: `https://www.dps-az.cz/zajimavosti/id:56560/bezdratove-technologie-pro-internet-veci`.
9. HAAG, Sebastian; ANDERL, Reiner. Digital twin – Proof of concept. *Manufacturing Letters*. 2018, roč. 15, s. 64–66. ISSN 2213-8463. Dostupné z DOI: `https://doi.org/10.1016/j.mfglet.2018.02.006`. Industry 4.0 and Smart Manufacturing.

10. *R4M#27: Co je to TESTBED?* [Online]. Praha: Elektrika.cz, 2017 [cit. 2021-04-13]. Dostupné z: https://elektrika.cz/data/clanky/r4m-27-co-je-to-testbed.

11. *Mobile Operating System Market Share Worldwide* [online]. 2021 [cit. 2021-03-12]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide.

12. LACKO, Ľuboslav. *Mistrovství - Android*. Brno: Computer Press, 2017. ISBN ISBN978-8025148754.

13. *Chapter 1. Android Overview* [online]. Sebastopol: Sebastopol, CA [cit. 2020-12-18]. Dostupné z: https://www.oreilly.com/library/view/learning-android-2nd/9781449336226/ch01.html.

14. MORRISSEY, Sean. History of Apple Mobile Devices. In: *iOS Forensic Analysis for iPhone, iPad, and iPod touch*. Berkeley, CA: Apress, 2010, s. 1–23. ISBN 978-1-4302-3343-5. Dostupné z DOI: 10.1007/978-1-4302-3343-5_1.

15. WINGET, Jenny. *A Complete And Informative Overview Of iOS Platform* [online]. Minnesota: EchoinnovateIT, 2012 [cit. 2020-12-18]. Dostupné z: https://echoinnovateit.com/ios-platform-overview/.

16. ALLEN, Sarah; GRAUPERA, Vidal; LUNDRIGAN, Lee. Windows Mobile. In: *Pro Smartphone Cross-Platform Development: iPhone, BlackBerry, Windows Mobile, and Android Development and Distribution*. Berkeley, CA: Apress, 2010, s. 65–80. ISBN 978-1-4302-2869-1. Dostupné z DOI: 10.1007/978-1-4302-2869-1_5.

17. *Windows 10 Mobile review: The future of Windows Phone is here* [online]. New York: Future US, 2020 [cit. 2020-12-18]. Dostupné z: https://www.techradar.com/reviews/phones/mobile-phones/windows-10-mobile-1286717/review.

18. *Application Fundamentals* [online]. Google [cit. 2021-04-13]. Dostupné z: https://developer.android.com/guide/components/fundamentals.

19. *Understand the Activity Lifecycle* [online]. Google, 2020 [cit. 2020-12-18]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.

20. *Services overview* [online]. Google [cit. 2020-12-18]. Dostupné z: https://developer.android.com/guide/components/services.

21. KROMPOLC, Tomáš. *Bluetooth je tu s námi 20 let: vše, co o této technologii potřebujete vědět* [online]. Praha 1: SMARTmania s.r.o., 2005 [cit. 2020-12-10]. Dostupné z: https://smartmania.cz/bluetooth-je-tu-s-nami-20-let-vse-co-o-teto-technologii-potrebujete-vedet/.

22. VYLEŤAL, Martin. *Bluetooth 4.0 neznamená konec předchozí verze* [online]. Praha: Internet Info, s.r.o., 1998 [cit. 2020-12-14]. Dostupné z: https://www.lupa.cz/clanky/bluetooth-4-0-neznamena-konec-predchozi-verze/.

23. *Bluetooth Protocol Stack* [online]. USA: Mathworks, 1994 [cit. 2020-12-18]. Dostupné z: https://www.mathworks.com/help/comm/ug/bluetooth-protocol-stack.html.

24. VYBÍRAL, Tomáš. *HLASOVACÍ ZAŘÍZENÍ NA BÁZI BLUETOOTH LOW ENERGY*. Brno, 2017. Bakalářská práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

25. TOWNSEND, Kevin. *Introduction to Bluetooth Low Energy: GATT* [online] [cit. 2020-12-18]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

26. *Bluetooth low energy* [online]. Mountain View, USA: Google, 2021 [cit. 2021-03-11]. Dostupné z: https://developer.android.com/guide/topics/connectivity/bluetooth-le.

27. *Privacy changes in Android 10: Some telephony, Bluetooth, Wi-Fi APIs require FINE location permission* [online]. Mountain View, USA: Google, 2021 [cit. 2021-03-11]. Dostupné z: https://developer.android.com/about/versions/10/privacy/changes%5C#location-telephony-bluetooth-wifi.

28. *VIZUÁLNÍ STYL*. Ostrava, [n.d.]. Dostupné také z: https://vizual.vsb.cz/cs/.

Příloha A

Zdrojový kód mobilní aplikace

Zdrojový kód mobilní aplikace je možné stáhnout ze vzdáleného repositáře na serveru GitHub (https://github.com/gawex/TestbedLogger). Zdrojový kód se rovněž nachází na přiloženém CD a v elektronické příloze pod názvem **A Zdrojový kód mobilní aplikace**.